Design, Development and Control of Mobile Biaxial Inverted Pendulum

by

Jonathan Missel

Friday, August 7^{th} 2009

A thesis submitted to the

Faculty of the Graduate School of the

State University of New York at Buffalo

in partial fulfillment of the requirements for the degree of

Master of Science

Department of Mechanical and Aerospace Engineering

Acknowledgements

First and foremost, I wish to thank my family for their unwavering support throughout my education. I have focused heavily on my education over the past several years, and they have all made sacrifices on my behalf. My parents, Dan and Linda Missel, have been exemplary models of success, both as parents and in their respective careers. Their dedication to selflessness has always been an inspiration to search for broader meaning. My dad gets special thanks for his many hours spent helping design and manufacture Vertigo, and my mom, for help editing this thesis. Thanks to my sister and her husband, Becky and Charlie Fennie, for their perpetual encouragement, sarcastic candor and comic relief. I would also like to extend my gratitude for their engenderment of my favorite quartet of playmates, Danny, Eli, Ainsley and Mara. Those kids are a blast, and their chaos is confoundingly soothing. I want to thank my brother, Matt Missel, for his support, companionship, ideas and advice as we grew up, and now pursue similar academic fields together. His mechanical wizardry was of tremendous assistance in designing Vertigo. Thanks to all of my friends back in Rochester for their understanding and encouragement as I have spent the majority of my academic career a city away.

Thanks to several in academia, particularly my advisor Dr. Puneet Singla, for being available and willing to help. It was his courage and insight that made gambling on my endeavors end in success. It has been a pleasure working with him, and I now

look forward to working under his former advisor as I continue on towards my Ph.D. at Texas A and M University. I would also like to thank John Wadach, Dr. Tarunraj Singh and Dr. David Forliti for their openness and inspiration on both personal and academic matters. Thanks to all of my labmates over the years, those in Dr. Singh's lab, Kurt Cavalieri and Jack Lee for weathering my antics with a smile, and maintaining an eagerness to assist. Special thanks to Shajan Thomas and Mark Tjersland for their computer genius and immense contribution as they volunteered their time and expertise to my research. I would like to thank cs for her prolonged belief, uplifting humor and studio photography skills. Lastly, thanks and apologies to anyone I may have inadvertently left out; I attribute this accomplishment to a myriad of combined efforts, and I thank you all!

Contents

Acknowledgements **iii**

List of Figures **xiii**

List of Tables **xiv**

Abstract **xv**

1 Introduction **1**

1.1 Motivation 1

1.2 Background 7

1.3 System Description 11

2 Design **17**

2.1 Introduction 17

2.2 Conception 18

2.2.1 Objective 18

2.2.2 Development 19

2.2.3 Design Challenges 21

2.3 Actuation Method 22

2.4 Components 28

2.4.1 Onboard Computer 29
2.4.2 Actuation 31
2.4.3 Sensing 33
2.4.4 Power 37
2.5 Structure 38
2.6 Design Iteration 48
2.7 Future Improvements 62

3 Mathematical Modeling and Analysis 66
3.1 Introduction 66
3.2 Mathematical Model 67
3.2.1 Introduction 67
3.2.2 Model Derivation 69
3.2.3 Linearization 73
3.2.4 Ground-Based Derivation 75
3.3 Analysis 77
3.3.1 Introduction 77
3.3.2 State Observability and Controllability Analysis 79
3.3.3 Design Parameter Observability and Controllability Analysis . 86
3.4 Conclusions 96

4 Communication Architecture 99
4.1 Introduction 99
4.2 Onboard Sensing Architecture 100
4.3 External Sensing Architecture 102

5 Control **110**

5.1 Introduction 110

5.2 Ground-Based Control 111

5.3 Sphere-Based Control 116

5.3.1 Power Optimal Control: *Fixed Final Time, Unconstrained Input* 117

5.3.2 Power Optimal Control: *Free Final Time, Constrained Input* . 120

5.3.3 LQR Overview 124

5.3.4 Stabilizing LQR 126

5.3.5 Tracking LQR 130

6 Estimation **137**

6.1 Introduction 137

6.2 Continuous-Discrete Extended Kalman Filter 138

6.3 Application of Continuous-Discrete EKF 140

6.4 Sampling Time Analysis for C-D EKF 148

7 Implementation **153**

7.1 Introduction 153

7.2 Implementation Challenges 154

7.3 Ground-Based Implementation 158

7.4 Balancing Implementation 169

8 Conclusions and Future Work **176**

Bibliography **181**

A Supplementary Concepts **184**

A.1 Introduction 184

A.2 Directionality 184
A.3 Translational Motion 186
A.4 Actuation 188
A.5 Control Superposition 190
A.6 Pivot Drift 192
A.7 Sphere Properties 195
A.8 Weight Distribution 196

B Powering Vertigo and Qwerk **198**

C Connection to Vertigo **201**

D Changing Qwerk's Embedded Code **214**

E Maintenance **237**

List of Figures

1.1 (a) Stationary manipulator (b) Statically stable vehicles (c) Omni-directional statically stable vehicle (d) Differential drive balancing vehicle (e) Bipedal walker. 2

1.2 Statically and dynamically stable vehicles encountering terrain. 5

1.3 Anatomical planes. 8

2.1 Fixed biaxial inverted pendulum concept. 20

2.2 Front, 45 deg. and top view of omni-directional sphere actuation concepts, (a) Concept A: inverse mouse, (b) Concept B: omni-wheel Euclidian perpendicularity, (c) Concept C: omni-wheel Euclidian orthogonality, (d) Concept D: omni-wheel spherical perpendicularity. 23

2.3 Vertigo 1.0 assembly photograph. 40

2.4 Vertigo 1.0 motor-encoder-wheel exploded subassembly (CAD). . . . 44

2.5 Vertigo 1.0 motor-encoder-wheel compressed subassembly (photograph). 45

2.6 Vertigo design generation. 50

2.7 CNC mill machining Delrin legs for Vertigo 2.1. 55

2.8 Modifying omni-wheels for Vertigo 2.1 use (photographs). 59

2.9 Vertigo 2.1 motor-encoder-wheel exploded subassembly (CAD). . . . 61

3.1 Planar sphere-based Vertigo model. 70

3.2 Ground-based free body diagram. 76
3.3 Rank of controllability matrix while varying θ and ϕ. 80
3.4 Rank of observability matrix while varying θ and ϕ. 81
3.5 Inverse condition number of controllability matrix while varying θ and ϕ. 82
3.6 Inverse condition number of observability matrix while varying θ and ϕ. 83
3.7 Determinant of controllability matrix while varying θ and ϕ. 84
3.8 Rank of controllability matrix while varying $\dot{\theta}$ and $\dot{\phi}$. 85
3.9 Rank of observability matrix while varying $\dot{\theta}$ and $\dot{\phi}$. 85
3.10 Rank of controllability matrix while varying $\dot{\theta}$ and $\dot{\phi}$. 88
3.11 Rank of observability matrix while varying $\dot{\theta}$ and $\dot{\phi}$. 88
3.12 Inverse condition number of controllability matrix while varying Vertigo mass and ℓ. 89
3.13 Inverse condition number of observability matrix while varying Vertigo mass and ℓ. 89
3.14 Determinant of controllability matrix while varying Vertigo mass and ℓ. 91
3.15 Inverse condition number of controllability matrix while varying Vertigo mass and ℓ with fixed rotational inertias. 91
3.16 Rank of controllability matrix while varying sphere mass and sphere radius. 93
3.17 Rank of observability matrix while varying sphere mass and sphere radius. 93
3.18 Inverse condition number of controllability matrix while varying sphere mass and sphere radius. 94
3.19 Inverse condition number of observability matrix while varying sphere mass and sphere radius. 94

3.20 Determinant of controllability matrix while varying sphere mass and sphere radius. 95
3.21 Inverse condition number of controllability matrix while varying sphere mass and sphere radius with fixed rotational inertias. 96

4.1 Onboard sensing communication architecture. 101
4.2 Ground-based communication schematic. 109

5.1 Ground-based free body diagram. 113
5.2 Simulation results for a circle reference trajectory: norm-error 0.7821 115
5.3 Planar sphere-based Vertigo model. 116
5.4 Gramian based control for 2π translation in 1.5 sec. 119
5.5 Gramian based control, u_max = 55. 121
5.6 Gramian based control, u_max = 45. 121
5.7 Gramian based control, u_max = 20. 122
5.8 Gramian based control, u_max = 15. 122
5.9 Plot of successful u_max convergences and associated final times. . . 123
5.10 State response with stabilizing LQR controller. 126
5.11 Response with stabilizing LQR controller wrt vertical. 128
5.12 State response with augmented stabilizing LQR controller. 129
5.13 Response with augmented stabilizing LQR controller wrt vertical. . . 129
5.14 State response with tracking LQR controller, heavily weighted rates. 132
5.15 Response with tracking LQR controller wrt vertical, heavily weighted rates. 132
5.16 State response with tracking LQR controller, heavily weighted angles. 133
5.17 Response with tracking LQR controller wrt vertical, heavily weighted angles. 133

5.18 State response with augmented tracking LQR controller, heavily weighted angles. 135
5.19 Response with augmented tracking LQR controller wrt vertical, heavily weighted angles. 135
5.20 State response with augmented tracking LQR controller, realistic weighting. 136
5.21 Response with augmented tracking LQR controller wrt vertical, realistic weighting. 136

6.1 Planar measurement model for sphere-based Vertigo. 141
6.2 C-D EKF, true state response. 145
6.3 C-D EKF, estimated state response. 145
6.4 C-D EKF, angle state errors. 147
6.5 C-D EKF, rate state errors. 147
6.6 C-D EKF, true state response, $f_m = 25Hz$, $f_s = 100Hz$. 149
6.7 C-D EKF, estimated state response, $f_m = 25Hz$, $f_s = 100Hz$. 149
6.8 C-D EKF, angle state errors, $f_m = 25Hz$, $f_s = 100Hz$. 150
6.9 C-D EKF, rate state errors, $f_m = 25Hz$, $f_s = 100Hz$. 150
6.10 Sampling time mesh. 152

7.1 Proportional two-dimensional step track. 160
7.2 Circle trajectory tracking using proportional controller with proportional yaw control: norm-error=2.3897. 162
7.3 Circle trajectory tracking using PID controller with proportional yaw control: norm-error=1.8976. 162
7.4 Circle trajectory tracking without yaw control: norm-error=0.8614. . 163

7.5 Circle trajectory tracking with proportional yaw control: norm-error=0.3866. 163
7.6 4 circle trajectory tracking laps with proportional yaw control. . . . 165
7.7 Circle trajectory tracking with transformed control: norm-error=0.5402. 166
7.8 Circle reference trajectory tracking while using proportional yaw control for various retrogrades. 167
7.9 Yaw control for 2π retrograde and 20π retrograde. 169
7.10 Balancing plank pitch with proportional control. 171
7.11 Balancing plank pitch with PID control. 173
7.12 Balancing plank pitch with PID control by tracking Vertigo. 173

A.1 Directionality (a) Bi-directionality (b) Bi-directionality with yaw (c) Omni-directionality (d) Omni-directionality with yaw. 185
A.2 Stages of translational motion. 187
A.3 Spherically orthogonal omni-directional actuation. 189
A.4 Ground-based trace of translation with rotation. 190
A.5 Inverted pendulum on cart. 193
A.6 Uncontrolled pivot drift. 194

D.1 Wiring for access to Qwerk. 217

List of Tables

2.1 Features of drive concepts. 27

2.2 Properties of Delrin. 41

Abstract

The limited versatility of existing experimental robotic platforms has left demand for a single practical unit with broad application in dynamics and controls. To address this, an autonomous biaxial robotic inverted pendulum is proposed that balances and navigates on top of a sphere. The spherical base and innovative actuation techniques permit omni-directional translation, and direct control of yaw. With this unprecedented motion, Vertigo, as it is called, is capable of modeling a myriad of other systems, and offers great potential for original work. To maximize versatility it was designed for reconfigurability, can accept spheres of different size and can operate on the ground as a more traditional omni-directional vehicle. This thesis focuses on the design, mathematical modeling, analysis, communication, control, estimation and early stages of implementation for Vertigo and its various configurations. Methods used in this development have general application, making the results and findings broadly relevant.

Chapter 1

Introduction

1.1 Motivation

The conforming nature of existing mobile robotic platforms has lead to an impressive understanding of like systems, but this convention has done little for the progression of unique or complex systems. By following this paradigm, designers have imposed significant limitations on their creations. The most severe of these constraints confine the maximum acceleration and deceleration, ability to withstand unexpected loading, and directional control. In the past these obstacles have been individually overcome with robots that either balance but only have two degrees of freedom, or have omni-directionality but are prone to become dynamically unstable. Recent attempts to completely liberate machines from such inabilities have led to the expensive and often unnecessary complications of bipedal walkers. These platforms have proved to be unreliable and well beyond most budgets of those interested in implementing them. For simplicity, many experimental platforms are designed to be linear, though non-linear systems dominate the real world. Linear control methods are only guaranteed to work on nonlinear systems over small realms, limiting the usefulness of these plat-

forms to test many practical applications. To form a genuine knowledge of dynamics and controls, there is a need to develop versatile robotics that can model a diverse range of other systems, and contribute original experiments without being overly convoluted or costly.

There are many desired characteristics for robotic platforms, they are mostly rooted in performance requirements and are challenging to satisfy wholly with a single unit. Specifications are dependent on objective; however, autonomy, exceptional mobility, responsive dynamics and cost effectiveness are among the most commonly sought after traits. Figure 1.1 shows a series of robots that give a rough idea of what is available.

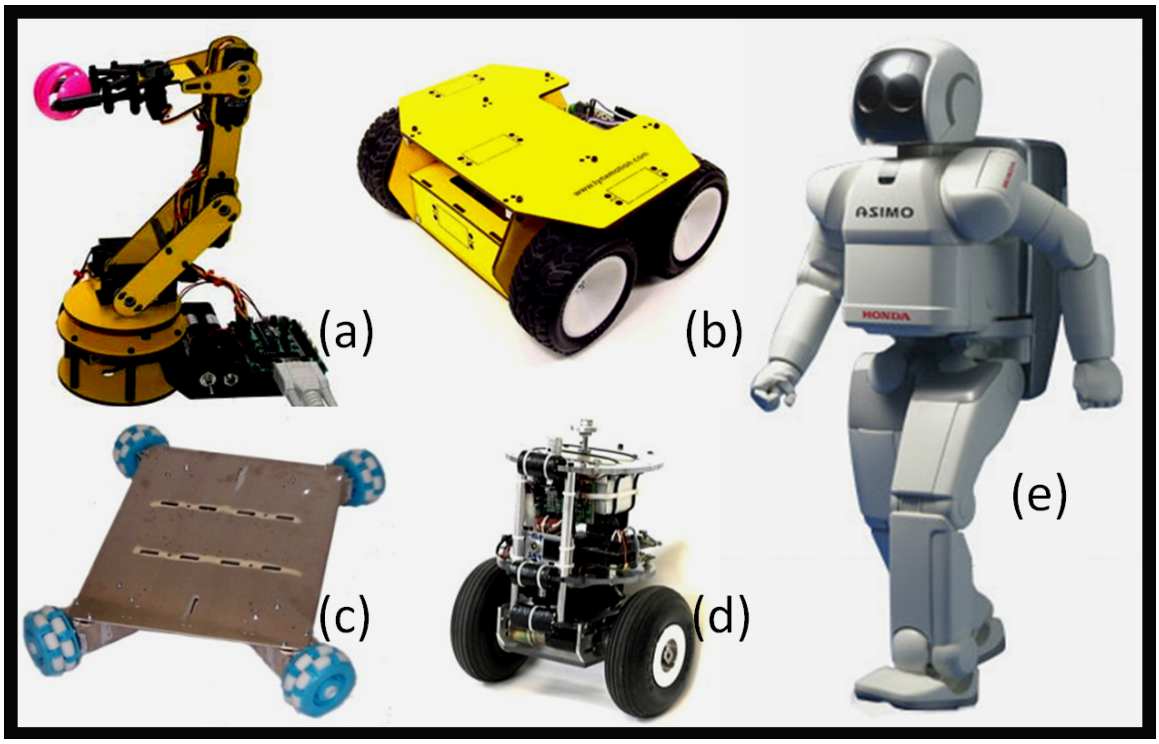


Figure 1.1: (a) Stationary manipulator (b) Statically stable vehicles (c) Omni-directional statically stable vehicle (d) Differential drive balancing vehicle (e) Bipedal walker.

Figure 1.1(a) shows a stationary robotic arm. These are useful in assembly applications, where their dexterity, accuracy and stamina often find them performing tasks that would be too mundane, difficult or tedious for humans. As with the numerous bench top robotics in existence, these are very precise and relatively easy to model, but as stationary robots their applications are limited. To provide mobility, statically stable vehicles, like the example in Figure 1.1(b), are often the default platform. To be statically (mechanically) stable simply means that on a flat surface the robot will not fall over under the acceleration of gravity when all control is terminated. These units are easily understood, and well documented which makes their implementation more straightforward; however, limits circumscribe their mobility. They commonly have four wheels where two steer similar to a car, or have differential drives like a wheelchair. In this way they only have two degrees of freedom because they can move forward/backward and turn, but cannot move side to side. Accordingly, the mobility of these systems leaves something to be desired, especially in close quarters. The acceleration of these platforms is also limited when their center of gravity is above the axles of their wheels. This is due to a likelihood of falling over. Similarly, extreme terrain can cause them to tip, or have wheels lift off the ground. Mitigating these issues requires a low center of gravity and large base, making them even more cumbersome.

Omni-directional vehicles, similar to the one in Figure 1.1(c), have made advances by solving some of the directionality limitations. These robots can move in any planar direction. Their rotation does not affect their direction of motion, so a complex path can be followed without having to spend time and power rotating. Further explanation of directionality can be found in Appendix A. Traditionally, these are also statically stable, and therefore inherit the risk of becoming dynamically unstable.

Balancing (dynamically stable) differential drive robots, also called self-balancing

robots, have begun to take advantage of dynamic stability. These are variations of inverted pendulums, as shown in Figure 1.1(d), and require unremitting control to balance because of their instability. However, this ostensible curse of instability is actually advantageous at times. Studies [1] have shown that dynamically stable robots are far superior at navigating terrain. This is because they are able to shift their weight to keep their center of gravity above their wheels, as shown in Figure 1.2 [2]. The value in this ability is most clearly seen when unexpected loading is applied and the controllers automatically compensate to maintain balance. Self-balancing robots also have greater potential for agility. Controlling where their center of gravity is allows dynamically stable robots to take advantage of their instability and let gravity assist in the task of acceleration. This is similar to how jet fighters are intentionally aerodynamically unstable for greater performance. In light of these dynamic prospects, the two-wheeled inverted pendulum configuration has been eagerly embraced as the focus of many research projects worldwide; however, this is a myopic response to straightforward results. Such vehicles do not take full advantage of the very feature that sets them apart, their dynamic stability. They are self-balancing on one axis, but maintain dependence with static stability on the other. Since they can still tip over in one direction, slopes must be dealt with head-on and directional control is lost. They also have problems with unexpected external loading applied in both directions, such as in the task of opening a door. Without prior knowledge of the exact size and location of the door, two-wheeled self-balancing robots cannot accomplish this. While they do have a niche, two-wheeled self-balancing robots can be viewed as an incomplete idea because they maintain the disadvantages of their constituents: they require active control to maintain stability, yet are still prone to tipping over.

Bipedal walkers like Honda's ASIMO (Figure 1.1(e)) have managed to exploit

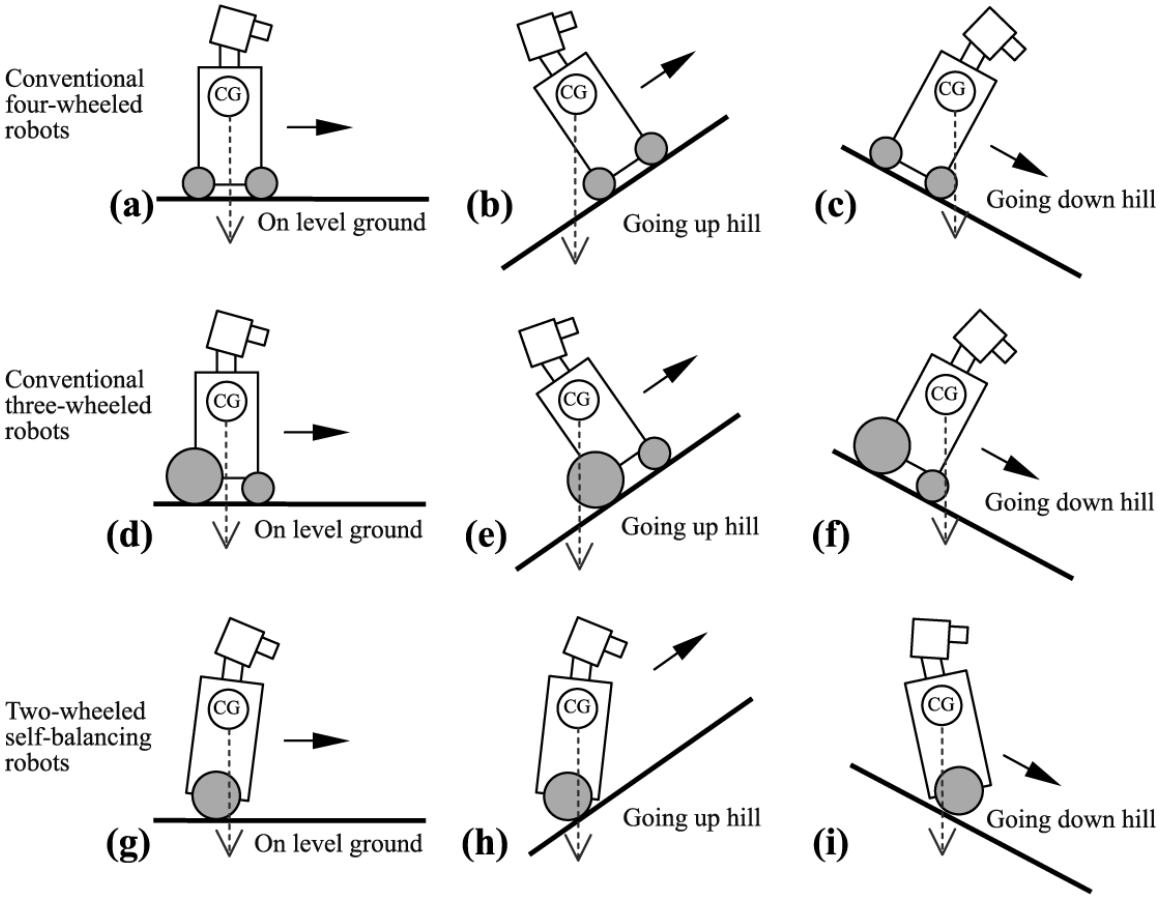


Figure 1.2: Statically and dynamically stable vehicles encountering terrain.

dynamic-stability in both directions, and are omni-directional to boot. These machines have taken advantage of biomechanical design by letting nature come up with the solution. Modeling robots after humans promises extremely agile performance; ASIMO can run, carry objects, and even climb stairs. With continued development, these platforms will even be able to outperform humans. However, all this grandeur comes at a tremendous cost, both fiscal and in robustness. Each of the 48 ASIMO units cost nearly $1,000,000 just to manufacture. This does not include the fund-

ing devoted to research and development, or any other aspect of its engenderment. Bipedal walkers are also inundated with single-point-failures which foster vulnerability and unreliability. ASIMO is 120 lbs of wires, actuators, sensors, circuits and structure, failure of any one of these components (with few exceptions) results in complete malfunction. The unreliability and cost of bipedal walkers has pushed their availability out of reach for most researchers, causing many facilities, such as MIT's Leg lab, to shut down.

The biaxial inverted pendulum was the iconoclastic idea that combined dynamic stability and omni-directionality in a frugal and simplistic manner. Rather than differential drive self-balancing robots, the conclusion that should have been drawn from understanding dynamic stability is to balance on a sphere and liberate both axes. By simply mirroring the effect of the two-wheeled inverted pendulum, this idea is not entirely disparate, and accordingly seems logical; however, an ongoing literature study revealed that this has only recently been done by one other group at Carnegie Mellon University with Ballbot in 2006 [3], [4]. The purpose of the Ballbot project is to investigate how to maintain reliable balance in robots to decrease their footprint for improved navigability. This is meant to overcome the problem of statically stable platforms having wide bases. Ballbot is a commendable project that has yielded impressive results with its mobility. However, there is one remaining feature that lingers, unaddressed: yaw control. Ballbot is omni-directional, so it can move and change direction without having to turn, but it also has no choice, it is not able to turn.

This thesis introduces a robot named Vertigo, a proposed solution to the final obstacle in achieving a dynamically stable system with all three planar degrees of freedom. As it was in the contributory stages, this evolution in mobility was an exercise in innovative mechanical design, and was rooted in punctuations of dynamics

and control theory. In many ways, Vertigo is similar to Ballbot, with its primary departure being the method of actuating the spherical base. It does so in a way that fully liberates all three forms of planar motion by directly controlling yaw, in addition to Ballbots two directional degrees of freedom. It was also designed to maximize adaptability by accepting different sized spheres, reconfiguring to change center of mass and inertial characteristics, and can even operate without the sphere as a statically stable omni-directional ground vehicle. This allows Vertigo to carry out maneuvers that no other vehicle can, giving it the flexibility to model and test control theory for a wide range of other systems.

Rocket guidance, under-actuated spacecraft, and in-flight refueling are among the challenging control problems that require extensive research and experimentation. Vertigo's mobility makes it capable of modeling these more expensive and complex systems which would help expedite their development in a cost effective manner. Vertigo also has the potential to make original contributions in the fields of rover locomotion, and human and machine cognition among others. This new form of mobility promises to affordably stretch the bounds of what ground vehicles are capable of.

1.2 Background

Recognizing the heritage from which Vertigo evolved is an important part of fully understanding it and appreciating the legacy that it carries on. The inherent instability of inverted pendulums has proved critical to their contribution toward technological advancement. They have been used in such diverse applications as seismograph instrumentation [5], biological neuron modeling [6], and testing control methodology. Although this configuration has been around for a long time, the development of

mobile inverted pendulums is in relative infancy. The geometry of these systems has a strong likeness to that of the human frame, for this reason, terminology from anatomy is frequently used. Figure 1.3 shows how the anatomical planes of the human body are defined. This chapter briefly outlines the background history of Vertigo's most unique feature, dynamic stability. More specifically, it covers the progression of mobile inverted pendulums and their directionality.

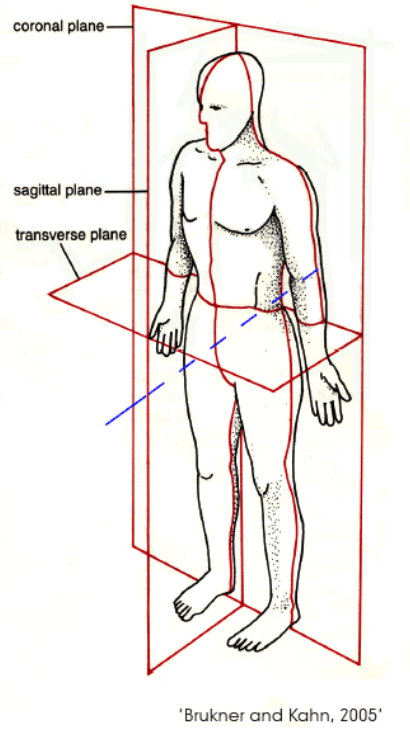


Figure 1.3: Anatomical planes.

As a robotic platform, this system got its start in 1986 when Professor Kazuo Yamafuji from the University of Electro-Communications in Tokyo successfully implemented the first wheeled inverted pendulum [7]. It was driven by a two-wheeled base, with both rotating in unison. In this way, it could not turn and is conceptually analogues to balancing on a cylinder. Lack of emphasis on documentation and pub-

lishing his work has brewed debate; however, this is regarded by many as the first autonomous inverted pendulum. In 1988, Professor Yamafuji came up with another, very different idea for the two-wheeled inverted pendulum [8]. An arm at the top of the pendulum was actuated to swing, providing a control torque that stabilized the pendulum. The two-wheeled base was controlled as if it were a statically stable device, and could turn like traditional differential drive vehicles. Since the wheels did not dictate the pendulum's motion, this device was really a fixed inverted pendulum with reaction wheel control for balance that was mounted on a mobile base.

In 1994, a group in Japan successfully demonstrated the control of the first two-wheeled, true inverted pendulum balancing robot that was able to turn. A differential drive allowed each wheel to be driven independently, enabling the base to rotate while its translational position governed the pendulum balance [9]. The ability to balance eliminated the need for differential drives to require a third, passive, caster that functioned solely as a static stabilizer. Later, they introduced the first one-wheeled balancing design. It rolled on the major axis of an ellipsoid to balance in the sagittal plane, and hinged at the "hips," to generate a control torque to balance in the coronal plane [10]. By balancing on an ellipsoid, the advantages of having a differential drive were lost, and rather than being able to turn in place, the robot was limited to wide arcs. This was also the first attempt at a mobile biaxial inverted pendulum; however, only one axis was controlled by the base, while the other was analogous to the reaction wheel control.

In 1999, inventor Dean Kamen started a company devoted to making clean transportation that focused on utilizing dynamic stability. The mobile inverted pendulum experienced its greatest boost in popularity in late 2001 when this burgeoning company unveiled the Segway Personal Transporter to the public for the first time [11]. It had a two-wheeled differential drive that utilized the same principles as its Japanese

predecessors, but its sensing and control systems differed. The most noteworthy aspect of the Segway is that it was designed to be a vehicle for humans, a very dynamic and unpredictable payload. The attractiveness of zero turning radius, speed, and the novelty of this design drew a lot of attention; however, it confined research to the two-wheeled differential drive configurations despite its many drawbacks. Their most significant shortcomings are coupling between yaw and directional control, and their tendency to become dynamically unstable when perturbed in the coronal plane.

Not until 2006 were the full benefits of dynamic stability exploited in ground robotics with Dr. Ralph Hollis' "Ballbot," created at Carnegie Mellon's Robotic Institute [3]. Ballbot is a true biaxial inverted pendulum that balances on a ball. Its chief advantage is its omni-directionality: it does not have to turn in order to change directions. It is also dynamically stable in both directions, so the same control methodology could be applied in both the sagittal and coronal plane. Ballbot was designed as a robot for researching motion in interactive human environments, so it took on the rough dimensions of the human frame. Its smooth motion has potential to assuage the resistance to assimilation of human and robotic cohabitation at home and in the work place. Though it is very slender, it is human height and difficult to transport, so testing requires a large space and is risky. This serves as evidence that it was not intended to be an experimental platform. Despite all its grandeur, Ballbot's configuration sacrifices direct control in yaw, a critical aspect of many trajectories. In addition, functionality is somewhat limited because it cannot be driven on the ground without the sphere, and the sphere has to satisfy very exacting dimensional tolerances. This is a result of the method used to actuate the sphere, which is similar to that of the ball and encoder assembly in an old computer mouse, but inverted. Two cylindrical rollers are used to drive the sphere in the x and y directions, but in order to prevent conflicting control inputs on the surface of the sphere they must

be perpendicular and placed on the equator (largest cross-sectional diameter). When one direction is controlled, the sphere rotates about the point of contact with the perpendicular roller. This method decouples directional control, but requires that the sphere be a very specific size to ensure the friction is sufficiently large enough to prevent slip, yet small enough avoid impeding motion. Since the actuation had to be placed at the equator, it was not able to support the weight of the robot on the sphere, and passive rollers had to be included in the design for this.

Though relatively new, the mobile inverted pendulum has quickly proven its usefulness. From its humble beginnings as a bidirectional bench top experiment, to its most recent manifestation as the biaxial, omni-directional and fully autonomous Ballbot, this platform has made rapid and fruitful progress. With this thesis providing a solution for its only remaining absence in motion, direct control of yaw, this platform awaits a strong future giving the robotics, dynamics and controls communities new tools for discovery.

1.3 System Description

This thesis proposes a solution to help solve the problem of limited mobility in robotics and its trade off with cost. The idea is that relatively inexpensive dynamically stable omni-directional motion can be achieved by developing a platform that balances, navigates and rotates on a single spherical wheel. This original design has potential for exceptional agility, and was developed to promote mass dimensional flexibility for adaptability to wide-ranging experimentation. As a fresh take on motion, it offers new avenues for research, and broadens the scope of ground vehicle navigation. Vertigo, as it is called, takes the form of a biaxial inverted pendulum with yaw control. The name was chosen for its blend of irony and accuracy. The word vertigo comes from

the Latin word vertigin, meaning "dizziness," and can be traced further back to the word verto, meaning "I turn." Today it is used as a medical term describing a form of dizziness that includes the sensation of spinning or swaying, often resulting in loss of balance. A symptom of balancing disorders seemed perfectly ironic for naming a robot whose party piece is its ability to balance. Additionally, the meaning "I turn," form which the word was originally derived, is quite fitting because Vertigo's ability to rotate as a biaxial inverted pendulum is what makes it unique.

Vertigo was designed to be simplistic. With only four actively moving parts, all of which are identical, the chance of mechanical failure is minimized; therefore, its simplicity translates in to robustness. With a height of only 17 inches, and weight of 5.6 lbs, it is easily transported, which is convenient for its role as a mobile experimental platform. The structural assembly was carefully designed with four legs that support two interchangeable mounting plates to allow reconfigurability and fortify its defenses against failure from impact. The combined influence of these traits granted Vertigo with longevity and the ability to contribute experiments that pertain to vastly different systems.

One of Vertigo's applications is modeling under-actuated systems; these are systems that do not have direct control of all feasible degrees of freedom. Fully exploring the dynamics of these systems is currently an active area of research; its mission is to discover and carry out maneuvers that control more degrees of freedom then are directly actuated. Requiring fewer actuators permits significant reduction in weight, cost, and energy needed to perform tasks. In addition, when fully actuated systems experience failure, these techniques may save the mission by using the remaining actuators to compensate for the loss. Laboratories in Japan are experimenting with passive joints to help improve the dexterity of under-actuated probes [12]. Similarly, Vertigo can be controlled to induce yaw rotation without direct control: a trajectory

can be assigned that combines pitch and roll for a resultant rotation in yaw. This maneuver is analogous to the way an airplane banks to turn, the rudder (primary yaw actuator) plays little role in carrying out substantial net changes in direction. By design, Vertigo 1.0 is under-actuated because it does not have direct control of yaw, so simulating this is straightforward. Vertigo 2.1 can also be configured or controlled to be under-actuated by only employing direct control over any two of the three degrees of freedom. With intelligent control, any combination of two of the four actuators can be eliminated, and the system can still secondarily dictate all three degrees of freedom.

Understanding how three degrees of freedom can be controlled by actuating only one is somewhat abstract for this system. However, it can be explained with the aid of a motorcycle stunt called a "circle wheelie," as an illustrative example. In this stunt, only the rear wheel of the motorcycle (used for propulsion) is on the ground, while the front wheel (normally used for steering) is suspended. While maintaining this stance, the bike and rider carve out circular trajectories by leaning to the side. This is an extremely technical stunt because the precise control of three degrees of freedom is completely dependent on the application of a single control input as the rider manipulates the throttle to power the back wheel and directly maintain pitch. When the bike naturally falls to one side, it turns in circles like a freely rolling coin. To avoid falling over, the lean angle (roll) must be kept in check. This is done through centripetal force by balancing speed with the roll angle while turning. To change the direction of the circular turns, increasing speed will cause the roll angle to surpass vertical and lean to the other side. Yaw is determined by how much of the turn is completed before straightening out. Speed of rotation can be increased by taking on a more aggressive roll angle with a shorter turning radius. To recap, pitch is directly controlled through powering the rear wheel, and the coupled motion of roll and yaw

is governed by centripetal force and how long the stunt is carried out for. This is a simplified explanation of the dynamics at work; phenomena like gyroscopic motion and momentum are not considered. Additionally, the coupling between translation and pitch is not stressed; when fully detailed, a total of four resultant degrees of freedom are being managed. Though simplified, this is a close example of how Vertigo 2.1 could remain operational with actuator failure. As long as any two are intact, it can be controlled analogously to a motorcycle on one powered wheel.

Even though it is possible to fully control an under-actuated system, it requires high precision and is more time consuming to execute than having a fully-actuated system; therefore, these methods are generally implemented when actuators fail unexpectedly, or when weight and cost make including more actuators unreasonable. This is often the case with spacecraft. In space, they are free to move in six degrees of freedom, but the cost of building and launching space vehicles makes reducing the number of actuators desirable when possible. Also, replacing failed actuators is rarely an option; therefore, these techniques may be exploited to save a mission using only what remains operational.

Vertigo is a highly maneuverable platform, and is capable of modeling more than just under-actuated systems. Some other examples with related motion to that of Vertigo are: the control of in-flight refueling systems, quad-rotor helicopter hover and planar translation, and actively controlled rockets. These systems are expensive to operate, risky to test and require highly trained personnel in controlled settings to ensure safety. Additionally, their development requires the progressive accomplishment of many stages that build up to a final result. A platform like Vertigo, that can mimic the general motion of these systems, is ideal for addressing some of the preliminary stages of development in an inexpensive, reusable and safe way. It can also be reconfigured to best match the simulated dynamic response, and its small size

and robust design make it practical for a single person to safely operate with minimal risk of failure.

As an experimental platform, applications of Vertigo are not limited to modeling other systems; it promises original contributions as well, particularly to the field of rover locomotion. As previously discussed, a biaxial inverted pendulum has exceptional performance and agility because it takes advantage of dynamic motion, rather than fighting it. With fully developed control, Vertigo can exploit omni-directional motion with independent yaw, scale inclines and terrain in any direction, and can actively adapt to unexpected loadings. These qualities make it ideal for accomplishing the objectives of many roving scenarios. Its fluid motion also has enormous potential to bridge the gap for human and robotic coexistence. Building this cooperation is a driving motivation behind Carnegie Mellon's Ballbot project. Smooth robotic interaction with humans mandates that machines have high centers of gravity and small footprints; however, this limits the maneuverability of statically stable robots due to their likelihood of tipping. Developing dexterous robotics may help the interface between lifeless machines and skeptic humans. In doing so, the importance of humanlike motion and dimensions is more than just psychological; cohabited environments expect both parties to operate in surroundings designed for the human frame. Though the setting may be different, this is the primary objective of NASA's Human and Robotic Exploration program, for designing robots that assist astronauts [13]. These machines must be built to work in environments meant for humans, without sacrificing functionality. The omni-directional motion and dynamic stability of Vertigo means that it is better suited than most platforms to work side-by-side with humans here on Earth.

Lastly Vertigo would make an exceptionally intriguing educational tool. There are hundreds of platforms and experiments on the market to help teach dynamics,

control and mechatronics, but most are limited in focus and fail to inspire students. Vertigo tends to invoke curiosity to those first introduced to it, and is flexible enough to accommodate work at all levels. Simple experiments can be set up to confirm and illustrate concepts taught in the classroom. Demonstrations that alter the physical properties and investigate the dynamics in comparison to theory would be an excellent aid for proving concepts at an introductory level, and would not be difficult provided a small bank of controllers. Additionally, its mobility and versatility offer much to be explored in advanced controls and estimation research.

Chapter 2

Design

2.1 Introduction

Designing hardware is a notoriously challenging aspect of experimentation. It requires thorough knowledge of how mechanical components fit and work together once assembled, and comprehension of underlying theory, so as not to corrupt the principles being tested. To simplify and insure the success of this process, previous work of others often serves as a foundation to build upon. Designing this particular hardware was especially challenging and critical because much of the projects originality stemmed from its mechanics. As a unique platform, there were few examples of pertinent work to guide its development; therefore, extra effort was made to ensure that functionality, accuracy and robustness were built into the design for its long-lasting success.

Many factors must be considered when conceiving and producing designs for a brand new robotic system. Often, these factors have conflicting solutions, and decisions must be made based on priority of tradeoffs. This chapter details how these decisions directed the conception, design and production of Vertigo. Starting with the

main objective, the evolution of the final product is detailed in full. Along the way, the challenges and logic behind every decision is explained, giving both the advantages and disadvantages of all considered options. The techniques and processes for manufacturing are also included, as this knowledge is an important tool for efficient design. Lastly, suggestions for improving or altering future designs are given.

2.2 Conception

This section discusses the engenderment of Vertigo as a concept. It starts by establishing the design objective, which is to be the motivation throughout, and the reason behind every decision made in the project's maturity. Various solutions to the given objective are then detailed, and supporting arguments are presented for a platform that balances on a sphere. Finally, the challenges implied by this solution are explained, and set to be addressed in the remaining design.

2.2.1 Objective

The primordial objective that seeded this project was to create a new ground platform with exceptional mobility, and mechanical simplicity. A machine capable of maneuvering with few constraints is attractive for experimentation and promises application in many areas. Achieving this in a cleverly simplistic manner, with no unnecessary parts, means that weight and cost can decrease while reliability and performance increase. Simplicity also aids in repair, assembly, disassembly, manufacturing and mathematically modeling. Therefore, the goal was to create an inexpensive testbed that was mechanically efficient, and had as many degrees of freedom as possible.

2.2.2 Development

To meet all aspects of the objective, a drive concept that provided mobility was first generated, and then an efficient design was created to embody this concept. Looking to the work of others, there were many existing testbeds that were highly maneuverable, but they were overly complex, or specialized in their application. Some were detached all together from any practical application, and served only as experimental amusement. Turning to the work of creation, it was clear that omni-directionality is very common in nature and uncontrolled systems. This would be very advantageous for improving mobility; however, it is notoriously difficult to accomplish properly in robotics. Nature also features many variations of the inverted pendulum throughout anatomical motion, and its purpose is often improved performance and agility. This can be observed when an agile creature like the cheetah turns while at speed. Front on, the animal carries the tall slender signature of an inverted pendulum. As it turns, it leans into the corner to cancel the centripetal force from turning. By matching the angle of lean with the aggressiveness of the turn, the animal experiences no lateral forces and can turn much sharper. A statically stable system in these circumstances would either tip over, or lose traction with the ground and slip. The inverted pendulum is also attractive because it is famous for being a benchmark problem in control. Accordingly, an omni-directional inverted pendulum was decided on for the desired design.

To make an omni-directional inverted pendulum, the bottom of the pendulum would ideally be actuated in both directions independently. The first concepts brainstormed for this were fixed platforms that supported the base of a pendulum and

actuated it in both directions. This was often done with perpendicular tracks or rails, like those in Figure 2.1. These devices could control the pendulum in any planar direction, but their motion was limited to the size of the chassis. A larger range of motion would require a larger device, which quickly becomes cumbersome and impractical. To overcome this restriction, a mobile vehicle would need to be used.

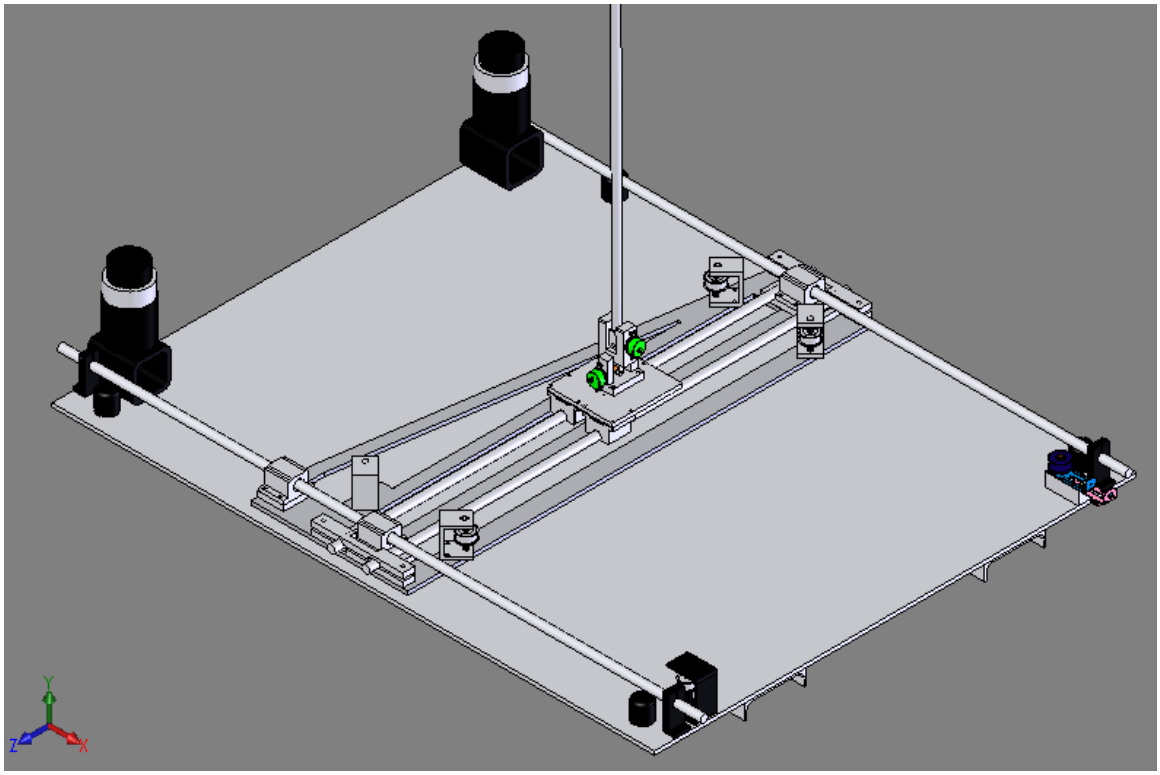


Figure 2.1: Fixed biaxial inverted pendulum concept.

Omni-directional vehicles had been done, but they were statically stable, with wide footprints and low centers of gravity. Inverted pendulum vehicles also existed, but they were not omni-directional; they could only move forward and backward and had to turn when changing directions. The idea of placing a pendulum on top of an omni-directional vehicle seemed promising at first, it would allow both features to be tested as controls experiments. This is similar in principal to the fixed platforms,

but would not be limited to the size of the base. The major drawback of this configuration is that it would not fully take advantage of the unstable dynamics of the pendulum. Cornering is still mostly dependant on the abilities of the statically stable base, with minor influence from the pendulum. Properly combining the two features would mandate a new platform all together. To ensure that the inverted pendulum dynamics were incorporated, it was necessary that the base only have one point of contact with the ground. This base also needed to move in any direction and hold as much symmetry as possible to maintain consistency in its dynamics. From these criteria, a sphere was chosen to act as a single-wheeled base that a pendulum would balance on.

2.2.3 Design Challenges

As with every new idea, there are many hurdles to overcome in the preliminary stages of design. With balancing on a sphere as the updated objective, the dominant struggle was developing a method for precisely controlling the sphere in two directions while it simultaneously serves as a base for supporting the robot. For control in any directions, multiple actuators must be interfacing the same surface while oriented in different directions.

Another challenge in the design of this robot, was maintaining the highest possible degree of symmetry to preserve the foreseen simplifications it would permit in the modeling and controls development. A symmetrical system should respond the same in all directions, yielding consistently predictable motion. To retain symmetry, all components (structure, electronics, actuators, etc.) must be designed, chosen and incorporated with precise intention.

2.3 Actuation Method

Pursuing such a unique drive made the method of actuation the most critical aspect to its success. Controlling one spherical surface in multiple directions is not a straightforward task because multiple actuators are required to be in contact with the sphere while avoiding conflicting control. By studying the geometry and motion of a sphere, four solutions to this problem were generated; Figure 2 shows the front, 45 degree and top view for the four concepts. The commonality in these methods is their omni-directional control, but there are many differences in their complexity, practicality and performance.

The first idea, concept A, came from contemplating the motion of a basketball spinning on a finger. If the finger is assumed to be a single point of contact, then the sphere can rotate about it, and the point will be uninfluenced by the rotation. To control the sphere's rotation, a second point of contact, a roller in this case, must be present. To avoid interference, the roller must be constrained to only apply control in the direction the sphere would naturally rotate about the first point. Now, if both points of contact are rollers, it can be deduced that omni-directionality can only be achieved if the axis of both rollers are in a plane that intersects the largest diameter of the sphere. Furthermore, control will be coupled for all orientations in that plane unless the two rollers are perpendicular. Since the sphere must be placed on the ground, a third point of uncontrolled contact is introduced. To account for this, the plane must be parallel with the ground, so the equator of the sphere is intersected by the plane. Figure 2.2(a) shows the geometry of this configuration. This is evidently the inverse of how an old computer mouse operates, when the ball rolls, perpendicular

rollers on the equator of the ball decompose the rotation and encoders are used to measure the component. Therefore, when used as an actuation method, only two actuators are needed and they independently control a component of the rotation.

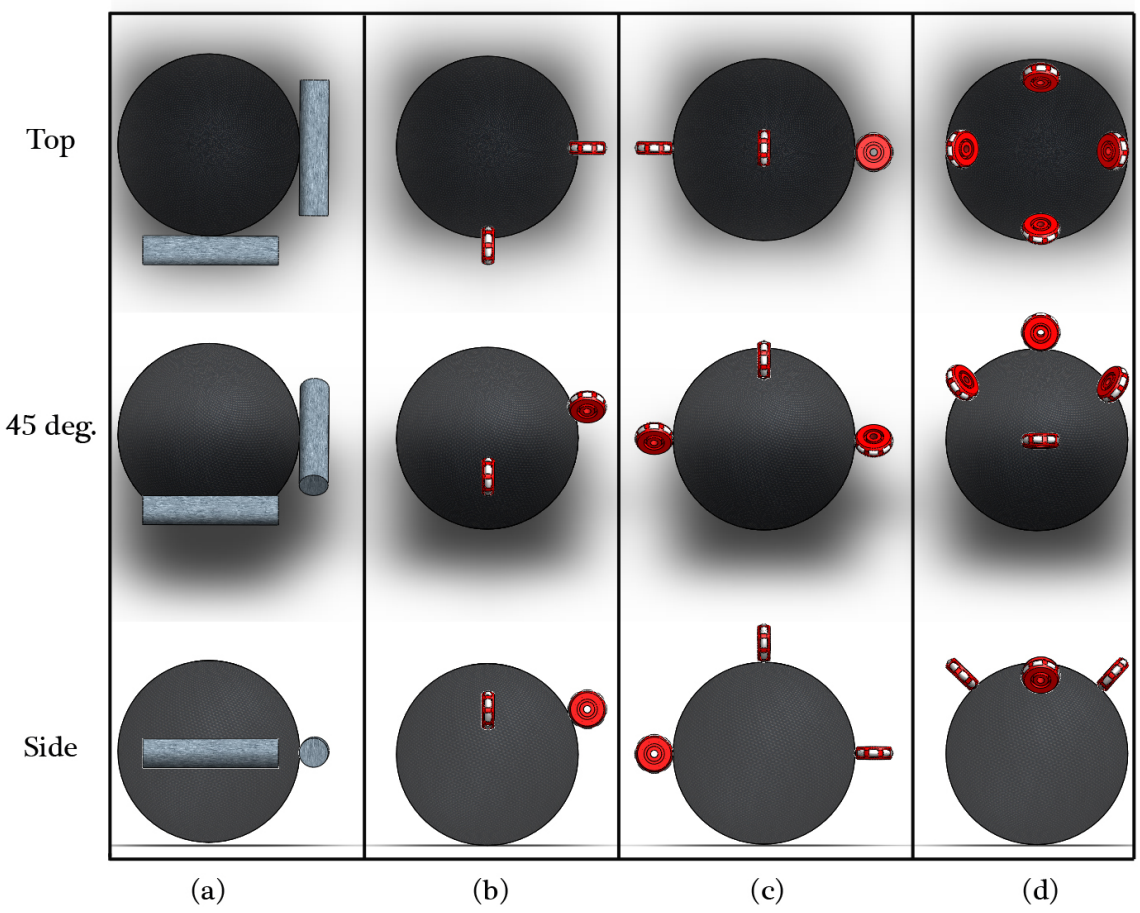


Figure 2.2: Front, 45 deg. and top view of omni-directional sphere actuation concepts, (a) Concept A: inverse mouse, (b) Concept B: omni-wheel Euclidian perpendicularity, (c) Concept C: omni-wheel Euclidian orthogonality, (d) Concept D: omni-wheel spherical perpendicularity.

Of the four designs, the inverse mouse drive is the easiest to understand, but there are several drawbacks. Though it only needs two actuators, it cannot directly control yaw rotation, so it only offers two degrees of freedom. The remaining drawbacks are attributable to the mandate of control implementation at the equator. Since

contact at the equator of a sphere does not support the weight of the vehicle above it, additional passive points of contact are needed to hold the robot. Additionally, just as it is with the computer mouse, at least one other point of contact must be placed on the sphere to apply force against the rollers. Lastly, this dependence on the sphere parameters means that each system has to be designed for a very specifically sized sphere. This is undesirable, especially when such trivial things as temperature and wear can change the diameter of the sphere enough to influence or incapacitate the system.

The second solution, concept B, came from a different train of thought all together. Traditional omni-directional robots overcome the problem of multiple actuators on the same surface with omni-wheels. Based on this, Figure 2.2(b) shows how this design's Euclidian perpendicularity can be adapted to conform to the contours of a sphere to give the same affect. As with the inverse mouse ball method, this has two actuators oriented perpendicular to each other and in a single plane that is parallel to the ground. However, by using omni-wheels rather than rollers, the actuator plane is no longer constrained to intersect the equator of the sphere. This is because all components of motion opposing the direction of control are passively permissible as slip, and the perpendicular orientation of the wheels decouples the directional control. Since control no longer needs to be applied at the equator, the sphere diameter is irrelevant as long as it is large enough to make contact with both actuators. This also means that the robot could be placed directly on the ground, analogues to a sphere of infinite radius. Being able to drive the platform using two, completely different methods is very attractive to the overall goal of experimental diversity and adaptability. This method does however have disadvantages. As with concept A, this needs passive contact with the sphere to help support the robot because the two actuators cannot do this alone. These two actuators each control their respective

degree of freedom and do not allow for direct control in yaw, which is not pertinent to omni-directionality, but would improve the mobility of the system. The remaining two concepts solve the problem of including yaw along with omni-directional actuation.

Keeping to the theme of one actuator per degree of freedom, Figure 2.2(c) shows how concept C proposes to completely decouple control in all three degrees of freedom by using three orthogonal actuators. Here, roll is controlled by an omni-wheel on the equator, pitch is controlled at the top of the sphere, and yaw is also controlled at the equator, exactly opposite to the one that controls roll. Two variations exist for this configuration, the yaw omni-wheel can be placed 90 degrees from where it is (either direction) along the equator of the sphere. These configurations put an actuator at each apex of a spherical quadrant, and orient them orthogonally in Euclidian space. The omni-wheel orientation is critical to this approach because each location depends on both the permitted slip from the wheels and the point of rotation idea from concept A to decouple control. Together, these three exhaustively compose the entire set of possible ways to independently control all three degrees of freedom using just three actuated omni-wheels. If the position or orientation of these actuators is off even slightly, control input will conflict and become coupled. Accordingly, this system is highly dependent on geometry, and will only work with a specific size sphere which is one of its flaws. Furthermore, this concept cannot operate on the ground. This is because it relies on the geometry of the sphere to locate points of rotation when decoupling control. Additionally, in all three configurations of concept C, passive contact with the sphere is needed to support the robot. In the previous methods, symmetry meant that control design was identical for both roll and pitch, so a planar model could be used to design a single controller, which could be applied to both axes. That is not the case here; all three degrees of freedom would require their own, slightly different, controller to account for their varied locations and lack of symmetry.

Concept D, shown in Figure 2.2(d), also includes yaw and omni-directional control. This method uses four spherically orthogonal omni-wheels, and is not dependant on the size of the sphere. Opposite pairs of actuators independently control roll and pitch, while a combined effort of all control yaw. This configuration also solves one of the shared problems of the other three that has not yet been mentioned. In every other method, control was designed to act through the axis of yaw rotation of the entire system; this passes through the sphere's point of contact with the ground and the center of mass of the robot. If the actuators are slightly misaligned, or the center of gravity is not at the geometric center of the robot (from a top view), or if the center of gravity shifts during operation, control will apply an additional undesired torque that acts to spin the robot and couple its motion. By jointly applying control from pairs of motors, input can be allocated to account for this if necessary.

All the previous concepts used one actuator per degree of freedom and are fittingly classified as equal-actuated; since four actuators are used to control three degrees of freedom, concept D is considered over-actuated. However, it is possible to use only three actuators, and eliminate the excess. With three, all of the omni-wheels axes of rotation would have to be oriented radially outward from the top view, and spherical orthogonality would no longer be required. The only constraint on the projected angle between the motors is that the sum of any two must be greater than 180 degrees, so this variation in itself can take on many forms. The sacrifice of only using three is that no control components are decoupled. This is true for all forms. For example, if the actuators all have equal projected angular spacing with respect to each other, then control is not applied in perpendicular directions. Even if two actuators are positioned perpendicularly, the control of the third motor would still be coupled. Additionally, yaw would be coupled with directional control because the perpendicular actuators would not apply control through the center of the sphere, causing a

resultant torque. With four actuators, only yaw is coupled, both directional control components are completely decoupled, allowing for the previously mentioned planar development. The major advantage of this over concept C, which also included yaw control, is that any sized sphere can be used here, and it can be placed directly on the ground while preserving all three planar degrees of freedom.

Concept	**# of Actuators**	**# of DOF**	**Sym.**	**Ctrl. Sym.**	**Self-Support**	**Omni-Wheel**	**Ground-Based**	**Any Sphere**
A	2	2	No	Yes	No	No	No	No
B	2	2	No	Yes	No	Yes	Yes	Yes
C	3	3	No	No	No	Yes	No	No
D	4	3	Yes	Yes	Yes	Yes	Yes	Yes

Table 2.1: Features of drive concepts.

Table 2.1 shows a comparison of significant features for the four drive concepts. The gravity of these statistics is subjective, and should be established to make an informed choice. To maximize versatility, a system that can accept any sized sphere and operate on the ground was desirable. This would also help with robustness against uncertainty in the sphere size. Recognizing that yaw control is not required for omni-directionality, and wanting to avoid over-actuation, concept B was initially chosen for Vertigo over concept D. To solve the support issue, two passive omni-wheels were arranged on the other side of the sphere; this also preserved the symmetry of the system, eliminating that flaw. As the project was under way, and concept B was under construction, a continued literature survey uncovered the recently published paper on Carnegie Mellon's Ballbot. This robot used the inverse computer mouse drive detailed by concept A. Although Ballbot required a precisely sized sphere and

could not be placed on the ground, it had the same degrees of freedom and number of actuators as Vertigo. A drive for complete originality pushed the project into an early design iteration that implemented concept D. This method had the same decoupled directional control as Ballbot, but had the additional advantage of yaw control and the ability to function on any size sphere or on the ground. This iteration came after the first version, Vertigo 1.0, was built, so the development of both designs is included in this chapter.

2.4 Components

With the objective and general method for accomplishment established, further development of the structure necessitated acquisition of all major components needed to make the platform functional. To determine what components were required, the various aspects of an implemented feedback control system were considered and found to be: sensing, actuation, computation and power. Sensing gives feedback measurements to the control loop so that the appropriate input can be determined. To carry out the control input, actuators are needed to influence the motion of the system. A computer is required to send and receive signals in accordance with the control loop, and possibly communicate with other computers. Finally, all of these processes need a power supply to run. These components had to be acquired before manufacturing the structure because its design was dependent on their size, weight and mounting. This section explains how the components were deemed necessary, the criteria they had to meet, and the specifications of the chosen items.

2.4.1 Onboard Computer

Intelligent robots need a brain to support their functions, and Vertigo is no exception. As the central hub of the control loop, the computer is charged with taking in sensory signals, determining the appropriate control, and generating an output signal for actuation. To determine the control, the computer must either apply its own embedded controller, or transmit signals to and from an external computer that does so. Therefore, the onboard computer should accommodate a broad range of inputs and outputs, and be able to communicate with other computers. Microcontrollers are capable of executing these tasks, but tend to be specialized, and are not easy to implement without experience. There are a handful small computers designed for general robotics use, and after careful consideration, Qwerk was chosen. Qwerk is a compact board that was developed at Carnegie Mellon's Robotic Institute and is produced by Charmed Labs. It is a powerful little computer with impressive specifications.

Qwerk specifications [14]:

- Qwerk Overview
  - Powerful robotics solution for university and hobbyist markets
  - High-performance CPU with an excellent I/O feature-set for robotics and mechinatronics applications
- Low-cost
- Hardware

  - 200 MHz ARM9 RISC processor with MMU and hardware floating point unit
  - 32 Mbytes SDRAM, 8 Mbytes flash memory
  - Linux 2.6 installed
  - WiFi wireless networking support
  - WebCam video input support
  - 4 Amp switching power supply, 90% efficient, 7 to 30 Volt input range
  - Rugged aluminum enclosure
  - 5.1" x 5.8" x 1.3", 11.8 oz

- I/O
  - 4 closed-loop 2.0 Amp motor controllers (supports quadrature encoder and back-EMF "sensorless" position feedback as well as current sensing)
  - 16 RC-servo controllers
  - 16 programmable digital I/Os
  - 12-bit analog inputs
  - 2 Rs-232 ports
  - I^2C ports
  - Built-in audio amplifier with MP3, PCM and WAV audio support
  - USB 2.0 host ports for connecting standard USB PC peripherals
  - 10/100BT Ethernet port

This computer was chosen over its alternatives for its dense package of features, and compact size. Qwerk met all desired specifications, and has additional features

that allow for future expansion and experimentation. It is able to accept a significant voltage range, thus keeping many powering options open, and its wireless networking interface is useful for stand-alone operation that will support autonomy.

2.4.2 Actuation

Much of the theoretical requirements for actuation were already covered by determining the drive method; however, selecting and building hardware to carry out these requirements has its own set of challenges. Choosing the omni-wheels alone had a great deal of tradeoffs. Due to the curvature of the wheel, gaps exist between the rollers. With existing wheels, a continuous control surface is achieved by having two rows of rollers, as seen in Figure 1.1(c) in the Motivation section. These rows are staggered such that the gaps of one row align with the rollers of the other. Therefore, on a flat surface these wheels will rotate smoothly. However, this application is controlling a sphere, and having two rows of rollers is not ideal because building a system that makes contact with both rows restricts it to a single sphere size, voiding one of the major advantages of this drive. Also, as the wheel rotates, and control application oscillates between the two rows, the output is inconsistent. Accordingly, a single row omni-wheel would have to be used for Vertigo. Unfortunately, the gaps in single row wheels make the affective radius inconsistent, and vibration from rolling on a discontinuous surface will add noise to the sensors.

In selecting single row omni-wheels, it was important that they have many rollers around the circumference to minimize the size of the gaps between them. The rollers should also be made of, or coated with, material that will provide traction on the various surfaces Vertigo will encounter. These criteria governed the verdict of choosing

Kornylak Transwheel 2051-10mmX CAT-TRAK wheels. CAT-TRAK is their synthetic rubber coating that is applied to the rollers for added grip and significantly improved performance.

Wheels specifications [15]:

- Standard 2" O.D. - 10mm Plain Bore
- Recommended max load:
  - Steel bottom = 25 lbs = 11.3 kg
  - Plywood surface = 7.5 lbs = 3.4 kg
  - 200 lb test corrugated bottom = 5 lbs = 2.25 kg
- Weight = 1.00 oz
- 8 rollers

The 10mm I.D. was attractive because it is a common size for hardware, which would simplify the process of finding fittings, and it was large enough to allow small hardware such as setscrews to fit within its diameter. Having eight rollers is dense for a 2 inch O.D. wheel, and the gaps are small because of this. Additionally, the roller axels are made from stainless steel, so rust will not be a concern, and their light weight will not appreciably hinder the response time of the motors.

The motors for Vertigo 1.0 were chosen primarily for their torque because this is the control input in the equations of motion. The motors also had to match the 7.2-12 volt range of Qwerk's output. Finally, Jameco Gear Head DC motors were chosen for their compatibility with Qwerk and because they had one of the highest

torque ratings in its class.

Motor specifications [16]:

- Rated voltage = 12 VDC
- Operating range = 6 - 24 VDC
- Current at maximum efficiency = 105 mA
- Speed at maximum efficiency = 8 rpm
- Torque at maximum efficiency = 3500 g-cm
- Gear ratio = 332:01:00

In Vertigo 1.0, passive contact is required for the drive mechanism; the same omni-wheels were also used for this. To let them spin freely, two 10mm bearings were press fit into their I.D.s. Spacers were made to position bearings flush with the outer faces of each wheel. The bearings inner races had an I.D. of 4mm, so 4mm partially threaded shoulder cap screws were used as axles, and threaded into brass slugs. These slugs were made to match the mass and diameter of the motors, and the cap screw axles were threaded in with the same offset as the motors driveshaft. All this attention to detail served to maintain symmetry throughout the system.

2.4.3 Sensing

The nature of this system mandates access to very accurate measurements of both the sphere and body dynamics for the feedback control loop. There are many types of sensors that measure attitude and acceleration, and in general, they tend to increase

in accuracy with price. Most inexpensive sensors are not accurate enough for this application. Some attitude sensors depend on the acceleration of gravity for their measurements, which is acceptable for static situations, but the dynamics of this system would invalidate their readings. An Inertial Measurement Unit (IMU) is a package of sensors that is specifically designed to measure the rates and acceleration of dynamic systems. Attitude is easily backed out from this information, so an IMU is well suited for this application. The IMU chosen was the Crossbow IMU400CD-100. It is a six-axis measurement system designed to measure linear acceleration along three orthogonal axes and rotation rates around three orthogonal axes. It uses three accelerometers and three angle rate sensors based on a 3-2-1 Euler angle system to make a complete measurement of the body dynamics. Methods such as dead reckoning can be used with this information to track the position.

IMU specifications [17]:

- Performance
  - Update rate > 100 Hz
  - Start-up time < 1 sec
- Angular rate
  - Rate range $= \pm 100$ °/sec
  - Rate bias $< \pm 1$ °/sec
  - Resolution $< .025$ °/sec
  - Bandwidth > 25 Hz
  - Random walk < 2.25 °/hr$^{1/2}$

- Acceleration
  - Range ± 4 g
  - Bias < ± .012 g
  - Resolution < .00006 g
  - Bandwidth > 75 Hz
  - Random walk < 1 m/s/ $hr^{1/2}$
- Operating temperature = -40 to + 71 °C
- Electrical
  - Voltage = 9 to 30 VDC
  - Current < 250 mA
  - Power consumption < 3 W
  - Output format: RS-232
- Weight < 1.4 lbs

From the specifications it is clear that this unit is extremely precise and does not take much power to operate. It also fits within the voltage limits of Qwerk and outputs the supported RS-232 data format, so it is more than adequate for measuring the dynamics of the body. Additionally, the physical dimensions are close to a cube, which helps maintain the symmetry assumptions in modeling.

For the sphere, the actuation method, detailed in the previous section, has effectively decomposed its rotation. Encoders that measure the rotation of the omniwheels would be able to provide information on the motion of the sphere with respect

to the body. Encoders are inexpensive, compact, and very common, so implementation is well documented. Quadrature encoders are particularly attractive because of their high precision, and they are supported by Qwerk. They have four slot patterns in their encoder disks and combine the readings from all to generate very accurate measurements. For this application, RE201 Kit Incremental encoders were chosen.

Quadrature encoder specifications [18]:

- 1 inch code disc
- Maximum line count of 4096
- 2 data channels in quadrature
- Standard 90° index pulse
- CMOS ASIC design
- 500 kHz frequency response
- 5 pin headers

In addition to these sensors, Qwerk supports electromotive force (EMF) feedback, which is a measure of energy per unit charge that the motor provides to the circuit. This is a function of rotational velocity for the motor, so EMF feedback is a sensorless way of measuring the motor rates. This method is not as precise as encoder measurements, but it can be combined with low weighting to improve overall accuracy. Qwerk also supports video feed via webcam. To verify which specific devices are compatible, the Terk website was consulted. Terk is an organization through Carnegie Mellon that provides support and advisement for the Qwerk unit, and was

referred to throughout this project [19]. From their list of webcams, the Logitech Communicate STX was selected for its performance. Not only is the inclusion of this webcam a means of video surveillance, but with proper image processing, it is high enough quality to back out attitude data. Further development of this could even lead to real time obstacle avoidance.

Webcam specifications [20]:

- High quality VGA sensor with RightLight^{TM} Technology
- Video capture: Up to 640 x 480
- Still image capture: Up to 1.3 megapixel with software enhancement
- Built-in microphone with RightSound^{TM} Technology
- Frame rate: 30 frames per second
- USB 2.0 certified

2.4.4 Power

Over the course of this project, Vertigo has been powered by a few different methods. Manufacturer defects made finding a suitable power supply that lasted challenging. The criteria for selecting a power supply were based on size, cost and the demands of the equipment. For wireless autonomy Vertigo needed to be battery powered, and lithium polymer (LiPo) batteries have the most attractive capacity, size and cost combination for this application. Based on the specifications of Qwerk and the other electronic components, it was determined that the power supply should provide roughly 12 VDC to the system. The amperage required varies greatly depending on

the usage, but an estimate of about 2.6 A/hr was determined from the component specifications. With this knowledge, packs of 2 Zippy-R 4150 mAh, 11.1 V three cell LiPo batteries were chosen. Wired in parallel with a Y connector, a pairs of these power bricks should last for more than 3 hours of constant operation. Two of these brick bundles were purchased so that a fully charged pair would always be on hand. They take about 1.5 hours each to recharge, so even if Vertigo is constantly in use, the other pack will have time to recharge.

A drawback of these batteries is that they can sustain damage if they drop below a minimum voltage. To address this, a voltage cutoff circuit was installed to terminate power when the battery provides less than 9 VDC. Another drawback to the battery is that they come in long brick shapes which, unless mounted vertically, do not promote symmetry. This would impose a product of inertia that couples the directional dynamics and causes departure from the planar model. To assuage this, the two batteries could be mounted perpendicular to each other, forming an X. This would improve the validity of the planar model by canceling the product of inertia.

Additionally, a computer power supply was hacked to provide a second option for powering Vertigo. It plugs into a standard outlet and provides 12 VDC at 1.6 A to Qwerk. This tether makes it more suitable for use when Vertigo is stationary, as it is for programming and diagnostics. Further details on powering options can be found in Appendix B.

2.5 Structure

With the majority of the components now specified, design of the structure could proceed. This section details the various aspects of the assembly design for Vertigo

1.0, with focus on the structural skeleton. For the purpose of discussion, Figure 2.3 shows a photograph of the assembled prototype. It will be referred to as the design is explained, starting from the top at the webcam, and finishing with the sphere at the bottom. The figure shows just one configuration of the assembly; variation will be explained in the discussion. Throughout the development, SolidWorks was used to design the robot in CAD; as components were ordered and received, the model was updated with exact measurements. The time and effort to make precise working CAD models is well worth the benefit; they are extremely convenient for design because the new ideas can be tried and analyzed, with time being the only cost. The model was used to fit all components and confirm that the design was feasibility before the final prints went to the machine shop. Renderings from this model are used interchangeably with actual photographs to help illustrate the various assemblies being discussed.

Perched on top of Vertigo is a very light weight plastic spherical webcam that includes a cover to protect the lens, and an image capture button on the side. It came with a mount for the sphere to snap in to, but it was designed to grasp computer monitors, and was not suitable for adhering to Vertigo. Therefore, a new mount had to be designed and built to withstand crashes, and possibly help protect the camera as well. The final mount design scavenged the clip from the original mount so the camera could be disconnected easily. This also protected the camera by allowing it to break free in collisions rather than bear the impulse of the entire robot. A ball and socket joint was incorporated into the design and handmade so that the camera could be prescribed to any desired fixed pan and tilt angle. This joint has .005 in of interference for the ball and socket which makes it stiff enough to maintain the desired orientation through small bumps, but gives it the flexibility to deflect in major collisions to discourage breakage of the mount or camera. Preventing deflection from small impacts was important for reducing uncertainty if the camera is to be used

Figure 2.3: Vertigo 1.0 assembly photograph.

for attitude sensing. The bottom of the mount was made to be flat, giving a secure contact surface for fastening with double sided tape. It was also center drilled to allow concentric mounting on a motor for actively controlling the pan. The metallic cylinder under the camera mount in Figure 3 is a small motor used to control pan; it has since been removed.

Continuing down the assembly, the yellow anodized aluminum cube is the IMU that was specified earlier. At each of its corners, a bumper was attached as a safety measure. These bumpers are marketed for child safety, and are intended to be fitted to the corners of furniture. They work perfectly for this application and their holes lined up with the screws in the top panel of the IMU, allowing them to be included without

modification. The electrical connection on the front face is composed of a scavenged DB-15 connector from a desktop computer, and was ideal for this assembly because its pins exited downward, protecting them from being bent. A 4-pin connector was then used to plug into the DB-15 outputs and cable into Qwerk. Attached to the side of the IMU is the USB extension and Airlink wireless adapter that allows Qwerk to communicate with other computers.

Further down is a black Delrin plate that the IMU attaches to. Delrin is a high grade engineering plastic whose properties are given in Table 2.2 [21]. In addition to these attractive characteristics it machines well, has a clean finish, is very durable, resists warping, resists corrosion, and is in the same general price range as alternate materials such as aluminum. For these reasons, Delrin was used for the entire skeletal structure and motor mounts.

Properties	**ASTM Test Method**	**Units**	**DELRIN**
Density	D792	lbs/cu in	0.0564
Specific Gravity	D792	g/cu cm.	1.56
Tensile Strength at 73° F	D638	psi	8,500
Linear Thermal Expansion Coeff.	D696	in./in./-°F	4.5 X 10-5

Table 2.2: Properties of Delrin.

The top plate is supported by four identical Delrin legs that link it to the bottom plate. The legs have cutouts to reduce excess weight and allow wires to be fed through them. The top and bottom holes in the legs that mount to the two plates are coaxial, and the hole patterns in both plates are identical so they can be interchanged for reconfigurability. All structural components are fastened with corrosion resistant stainless steel #8-32 socket head cap screws so it can be completely disassembled

with only one tool. These cap screws are recessed in counterbored holes so they seat flush; this maximizes the usable mounting surface area on the plates, and maintains the clean lines of the skeleton. The bottom plate holds the battery pack which is attached via hook-and-loop so it can easily be attached, removed, and relocated. On the underside of the bottom plate is Qwerk, the onboard computer. To further promote reconfigurability, mounting holes were made in Qwerk that mirrored those of the IMU. Also, the holes in the top and bottom plates were completely threaded through; this allowed the IMU to be mounted on either side of both plates, and Qwerk on the top of the top plate, or the bottom of the bottom plate. This is true for all structural configurations. The larger plate was sized to match the dimensions of Qwerk to help protect it, and fit the curves of the legs in the shown configurations. To further protect Qwerk, it was sent out to be anodized black; this reduces pitting from corrosion on its aluminum casing.

In Vertigo 1.0, the legs not only connect the plates, but also serve as motor mounts. For simplicity, this was all accomplished in one solid piece, so only six structural components assemble to form the skeleton, four of which are the identical legs. The drawback to combining the legs and motor mounts into one part is that the wheels had to be offset from the center of the sphere. It was understood that this would influence the motion slightly in practice, but theoretically it was not very significant. The dynamics of this system ideally prefer control along the axis of symmetry and through the center of gravity. To accomplish this, the structure would need to be off set to accommodate the wheels. This would further weaken the assumption of symmetrical inertias. Since the omni-directional wheels always act through the center of the ball anyway, this is physically similar to having the legs in the plane of symmetry and offsetting the wheels. Therefore, straight legs were just as affective and were made to greatly simplifying their design and manufacturing. The biggest repercussion of

straight legs is that the body will, in affect, be rotated with respect to the reference frame that control is applied in, slightly coupling the dynamics. The lower portion of the leg was designed to permit some flex to help all four wheels maintain contact with the sphere, functioning similarly to the suspension of a car. Additionally, permitting flex absorbs some of the shock from impact, which helps to protect the robot as a whole. As an adaptable vehicle, determining exact specifications for this permitted flexing was impossible because it required knowledge of the mass and inertia which are variable. Therefore, from experience with the material, the legs were designed to allow a nonspecific midrange amount of deflection, but were kept rigid enough to avoid appreciable departure from the rigid body assumptions in the mathematical modeling. In practice, this design proved to be successful.

The motor mounts were designed specifically to clamp tightly down on these particular motors. Accordingly, the diameters of the ballast motor blanks that support the two freely spinning wheels were sized to that of the motors, and their length was set to match the motors weight. The blanks accepted a 4 mm shoulder screw that functioned as an axel for the bearings that press fit into the wheels. A Delrin spacer was made to fit between the bearings and keep them flush with the outside surfaces of the wheels to help them run true.

The motor-encoder-wheel assembly was challenging to design. The encoder wheels were very fragile, and the assembly had to be as compact as possible for minimal compromise of symmetry due to wheel offset. For the encoders to function properly, the reader circuit had to be fixed with respect to the motor, and the encoder wheel must rotate with the omni-wheel. Both wheels had to be fixed to the motor shaft, and the encoder wheel had to line up with the optics slot in the circuit. Figure 2.4 shows the exploded CAD model of the final design solution for this assembly. The key components to this design are the wheel hub and the encoder circuit mounting

plate. The aluminum mounting plate screwed into preexisting holes on the face of the motor. This would fix the circuit and gave the assembly a solid and consistent stop for securing in the motor mounts. A hole in the plate allowed the driveshaft to pass through the encoder assembly. The encoder circuit screwed into the thin plate and was positioned such that the wiring harness plugged in along the side of the leg to shield it from impacts.

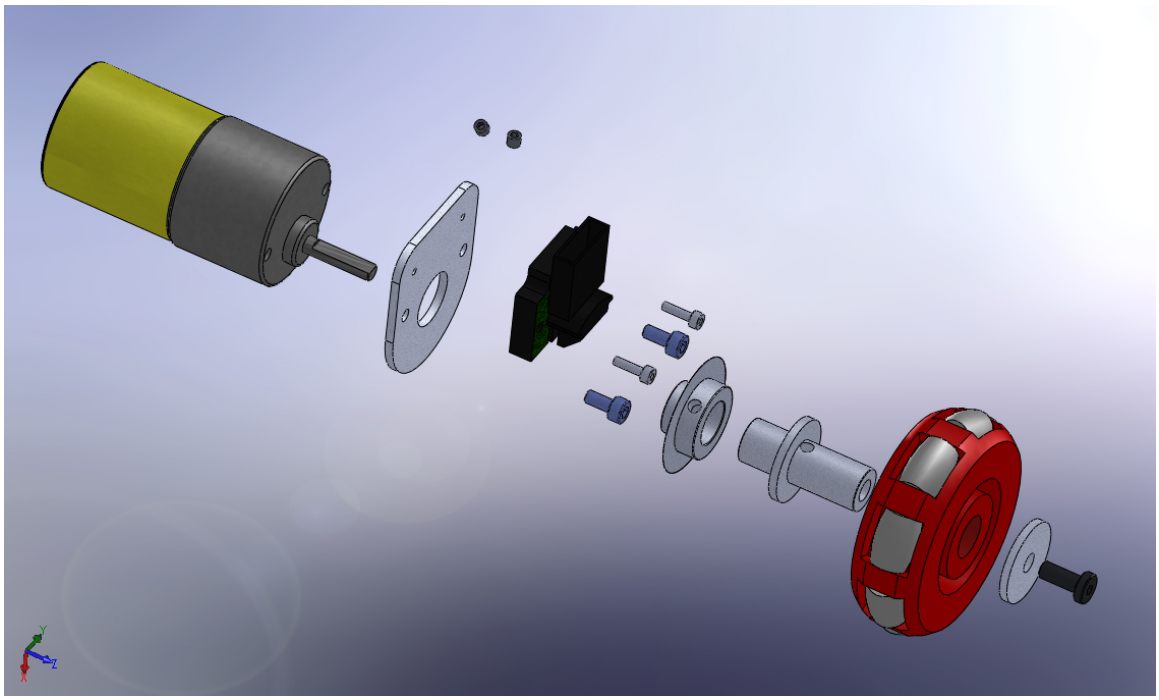


Figure 2.4: Vertigo 1.0 motor-encoder-wheel exploded subassembly (CAD).

The encoder and omni-wheels were secured to the driveshaft with an aluminum hub. The hub was fitted with a setscrew that tightened on the flat of the shaft. The inner half of the hub accepted the encoder wheel, which was secured with a setscrew. The hub was designed such that when it was seated at the bottom of the shaft, the encoder wheel lined up perfectly with its slot. The hub was given a stop to consistently locate both wheels from their respective ends, and allow clearance between the omni-

wheel and encoder plate. The omni-wheel also needed to be fastened to the hub, but its geometry was not ideal for fitting a setscrew. To address this, two mechanisms were utilized for securing the wheel. For the first, the hub radius was made with .002 in of interference with the wheel I.D., forming a snug press fit. Secondly, this section was made .005 in shorter than the thickness of the wheel, and threaded at the end so that a screw and washer could clamp the wheel against the hub stop. Though it increased the offset slightly, the encoder was mounted between the motor and the omni-wheel to protect its delicate exposed wheel. For all of its intricacy, the final design was very compact, as seen from the photograph of the compressed assembly in Figure 2.5. As a final modification, flats had to be milled between the rollers of the wheels to allow clearance with the curved surface of the sphere.

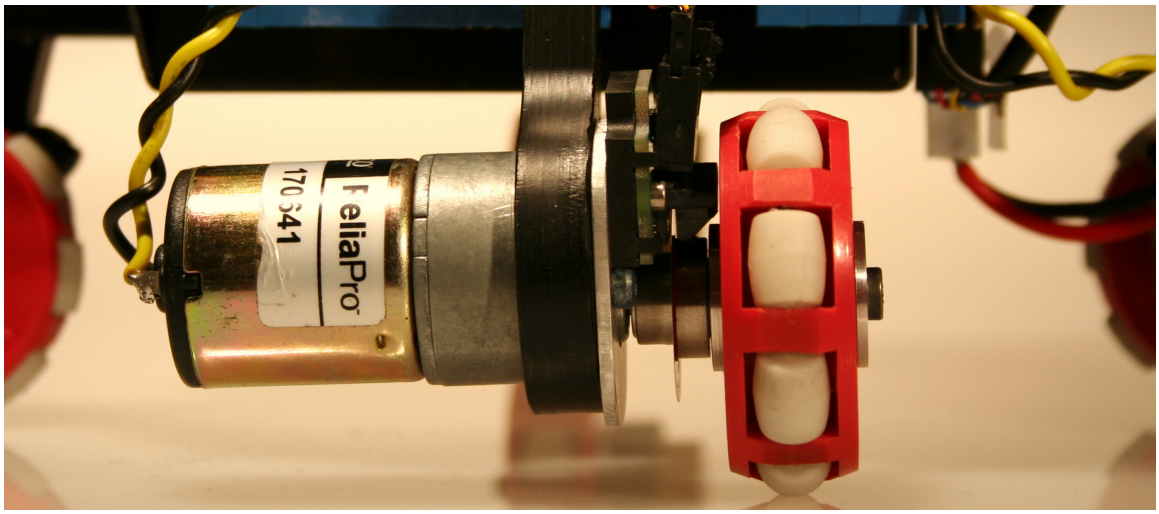


Figure 2.5: Vertigo 1.0 motor-encoder-wheel compressed subassembly (photograph).

Since the conception of this project, much thought and debate was directed toward determining the ideal sphere for the robot to balance on. Though it was intended to fit any size sphere, a choice had to be made for acquiring the first one. The design controllability and observability analysis shed light on how this decision could be made

to simplify the problem, but it was understood that this was an experimental platform, so a degree of challenge was welcomed. It was concluded that a hollow sphere would minimize rotational inertia and increase agility, and that a rubberized finish would mitigate concerns for traction with both the omni-wheels and ground. Eventually, a basketball was settled upon to launch the project because they satisfied all these requirements, are inexpensive, come in standard sizes and are readily available. Once acquired, the surface was lightly sanded to reduce the severity of the bumps and groves in the ball.

In general, there were a few critical notions that guided this structural design: simplicity, durability, reconfigurability, symmetry and manufacturability. The simplistic and open structure was intended to make assembly and repair straight forward. This also gave fewer parts the opportunity to fail, and made visually analyzing problems easier. Durability was secured by designing a compact and rigid skeleton made of Delrin. Safety features like the camera mount, IMU bumpers, Qwerk mount and encoder assembly also help protect the robot and improve overall hardware robustness. The primary service that reconfigurability provided was the ability to easily change the center of mass and inertial properties of the robot without the use of dead weight. As an experimental vehicle, it was important that the various constituents be flexible to test which setup produced the best results and to make room for additional equipment. For this reason Qwerk was modified to accept the same mounting hole pattern as the IMU. Also, the two plates were designed to be interchangeable, with universal mounting hole patterns on both sides. This allows Vertigo 1.0 to be assembled in 12 unique configurations of the major components, each with different center of mass and inertia. The design was also adaptable in that it had many mounting surfaces to attach new equipment as needed. In any given assembly, one side of a plate is completely empty, and the IMU and Qwerk were left exposed to serve as additional

mounting surfaces.

Symmetry was stressed to permit simplifying assumptions in the mathematical modeling. This was achieved by designing and purchasing symmetrical components, and assembling them such that coronal and sagittal planes had optimally similar inertias. The major departure from this was in the motor mount and assembly; this offset the wheels, but was not severe and was consistent for all four. Therefore, the angle could be determined and accounted for in the dynamics if deemed necessary. Having the privilege of designing and building a system to best fit its equations of motion was unusual, but welcomed.

Lastly, manufacturability and assembly was at the forefront of design for every part. It is very common that engineers design parts to meet their mechanical objective in an assembly, but their prints get sent back from the machine shop because they are infeasible or impractical to manufacture. By understanding the processes used in manufacturing the parts, cost, time and consistency can be improved with intentioned design. All of Vertigo 1.0's structural components were cut out on a CNC mill with a 1/8 in end mill. Therefore, 2-D designs were used to simplify production, and all inner radii were limited to 1/16 in. Flat, or 2-D components are easier and less expensive to design and manufacture because they only require that one two-dimensional profile be sent to a CNC mill which can complete the entire cut in a single tool-path.

Knowledge of the manufacturing procedure was used in aesthetic design as well. To create tool-paths for the mill to execute, the SolidWorks prints had to be exported into MasterCAM, a computer aided machining program that can write programs for the mill. In MasterCAM, splines are reduced from equations to a series of representative tangential lines. There is a fixed number of lines per spline, so short or conservative curves appear to be smooth, but long profiles do not and distract from the overall appearance. This is because light catches the flats like a disco ball, and it is easy to

see that the shape is discontinuous. For this reason, the dramatic swooping curves of Vertigo's skeleton were composed of combinations of tangentially joined fixed-radius arcs whose equations are readily passed between the programs and give a smooth finish to the profile.

The method of assembly also had to be accounted for in the design of the encoder mounts. The initial design was feasible to manufacture, and fit properly when assembled, but analysis of the CAD model revealed that it was not actually possible to assemble. Fortunately, this was spotted in the early stages of design, before manufacturing, and served as a testament to the advantages of maintaining an accurate and up-to-date CAD model.

2.6 Design Iteration

Before moving on to the Vertigo 2.1 design iteration, a quick recap of the project's evolution to this point is given. The discussion is complemented by the images in Figure 2.6 which illustrate some of the major phases in Vertigo's design generation. Figure 2.6(a) shows the first CAD rendering that depicted the conceptual fundamentals and a primitive vision for the design. It established the idea of using an inverted pendulum and balancing on a sphere, it even began to work out the orientation of the actuators for omni-directionality. This sketch launched the project by helping to explain its main ideas.

Once the project was accepted, the model in Figure 2.6(b) was created to help determine what components were needed, and how they might be assembled. This model established the actuation method and sensing equipment to be used by includ-

ing components that were dimensioned based on the published specifications of the manufacturer. At this stage, early ideas for efficient design and reconfigurability were also being explored. As the components were ordered, and the design took shape, the continuously updated working model evolved into that of Figure 2.6(c.1). This included all components of Vertigo 1.0, modeled precisely to measurements. Most importantly, this was the final design for Vertigo 1.0 that was analyzed and approved for manufacturing. Figure 2.6(c.2) shows a photograph of the physical system after manufacturing. Side-by-side with the CAD rendering, it is apparent how accurate the model was, and why it could be used with such confidence for designing the structural components.

Figure 2.6: Vertigo design generation.

After Vertigo 1.0 was designed and manufactured, the early stages of implementing simple feed forward control highlighted some of its shortcomings. When operating on the ground for extended periods of time, friction from the offset wheels tended to result in a net rotation. Also, the motors were too slow in practice, even after modifying the gear box to eliminate one of the gear reductions. It was decided that these problems were not debilitating for preliminary testing and experiments, and fixing them could be delayed until necessary. It was at this time Carnegie Mellon's Ballbot paper was encountered. Ballbot did not work on exactly the same principles as Vertigo 1.0, but it was very similar. This pushed the project into an early design iteration to make use of drive method D for actuating the sphere. This would give Vertigo direct control in yaw, which Ballbot did not have. It would also eliminate the possibility of motor offset interfering with the dynamics of the system. Since both the motor offset issue and new drive method only pertained to the legs, the remainder of Vertigo 1.0, for which the design proved to be very successful, was preserved. This reduced the time and cost of redesign, and still allowed the Vertigo 1.0 legs to be fitted if desired.

Vertigo 2.0 and 2.1 differ only in that new actuators and motor mounts were fitted to Vertigo 2.1. With every other aspect redundant, only the design of Vertigo 2.1 is included here. As with the explanation of the original design, discussion of the redesign is accompanied by visuals to better illustrate ideas. Figures 2.6(d.1) and 2.6(d.2) are respectively the CAD model and photograph of the completed Vertigo 2.1 design. Features will again be explained from the top down, starting with the first modification at the top of the legs.

The spines that protrude vertically from the tops of the legs were included to give further protection to the components on top of the upper plate. They were specifically sized to shield the wireless USB adapter, the IMU connector and the

retro-reflective markers used for tracking with Vicon (see Chapter 4). Further down the legs, larger cutouts were included to make wrapping and securing wires easier. Attached to the underside of the top plate is a speaker which makes use of Qwerk's audio compatibility. This was initially included for amusement, but proved to be very useful in debugging code, as different tones were scattered throughout implemented programs to pinpoint errors.

To fortify protection in the highly chaotic early stages of implementation, a pneumatic rubber bumper (not shown) was stretched around the narrowest span of the legs. It consisted of a nine inch lawnmower inner tube, inflated to half capacity. Rubber construction made it durable to resist puncture, and with its volume primarily occupied by air, its mass and inertia did little to affect the system's dynamics. The bumper provided an inexpensive and affective buffer that cushioned most components from impact.

Continuing down the robot, the next major design change was the pronounced kinks in the legs. These were included for several reasons. In Vertigo 1.0, accessing Qwerk's ports was somewhat difficult. The USB ports were particularly tight, and the plastic casings on the plugs had to be shaved down. Increasing the distance between Qwerk and the legs not only freed up all of the electrical connections, but it also protected the USB peripherals, whose plug ends were exposed in the original design. The exact distance the legs extended was determined by the overall assembled geometry of Vertigo. Together with the upper spines, the top plate, and the bottom plate, these elbows were spaced so that at no point when Vertigo 2.1 is on its side will the IMU or webcam contact the ground. This is not to say that these components will never hit the ground if Vertigo falls off the sphere, or tipping over at speed, but it is very affective at protecting them from minor falls. It is also beneficial for working on Qwerk, allowing the body to safely rest on its side. Lastly, the angle of the elbow was

determined in accordance with the full leg assembly. Excluding the motor mounts, the new legs had to be made in two parts for manufacturing reasons. The holes that mount the bottom plate had to be drilled and tapped from the underside of the legs. This meant that the mill had to have clearance to maneuver, and the lower portion of the legs would interfere if made of one solid piece. Therefore, the legs were designed in two parts and the angle of the kink was assigned for ease of assembly and to form a sturdy perpendicular joint. As before, all skeletal joints are fastened with the same size screws to make assembly, reconfiguration and maintenance easier.

Making up the bottom half of this joint is the lower leg. Its function is to recover from the wide conclusion of the upper leg, and locate the actuators. One of the convenient aspects of having two-part legs is that if the desired angle of the actuators changes, only the bottom section needs to be redesigned. This saves material cost and time. In this case, the lower legs were designed to position the actuators sufficiently far below Qwerk so as not to limit access to its electrical connections or prevent it from being removed without disassembling the legs, but not so far down that the legs were prone to vibration. Note that, similar to the original legs, these were designed to permit deflection for improved contact with the sphere, and absorbing impact. However, this was done in moderation to maintain the rigid body assumption.

Another concern that was solved by the lower legs was that colliding with walls in ground-based mode could damage or dislodge the actuators if they bore the brunt of the impact. For this reason, the lower legs tuck the actuators under such that the sturdy leg knuckles impact first. In practice, this was also found to be true when operating with the sphere; more often than not, the rotation from tipping over had Vertigo impacting the ground horizontally, and the knuckles protected all other components from damage by impacting first. The angle of the lower leg's bottom face determines the actuator's angle of attack. These were designed to orient the

omni-wheels near perpendicular with respect to the surface of this sphere; however, this was not mandatory. Since the omni-wheels always act through the center of the sphere, making contact at an angle is simply analogues to running on different sized wheels. Accounting for this in the controller gains or equations of motion can easily be done experimentally or geometrically. Designs for adjustable legs were generated for consideration, but they were decided against due to their complexity, and durability concerns. An adjustable joint is not as secure, and jarring from impact could easily misalign them. This is not only inconvenient, but would make fixing all four joints at exactly the same angle difficult.

Fusing motor mounts into the legs was no longer practical with the redesigned actuation method. The new mounts had to secure the driveshafts of the motors in the plane of the legs. Logically, this meant that the motors should be fixed to the bottom of the legs, making Vertigo 2.1's mounts vastly different than the originals. In the final design, two clamps secured each motor, and were made specifically to fit the motor diameters. Therefore, if new motors were ever used, only the clamps had to be remade. Each clamp connected to the leg with only one bolt which permitted them to pivot in the absence of a motor, but with a motor, their motion was fully constrained. These particular motors changed diameter along their length; therefore, the two clamps had to be made different sizes. They suspended the motors such that, when assembled, the bottoms of the legs made perfect tangential contact with the largest diameter of the motor. This formed a very stiff assembly, and transferred most of the pressure away from the mounts; therefore, they did not have to support the weight of the body which allowed them to be smaller and lighter.

To give the design some flexibility, the motors could be adjusted by sliding them further in or out of the clamps. This let the posture of Vertigo be very quickly altered to raise or lower the center of gravity, and take on a wider or narrower stance.

If desired, the motors could even be completely reversed, keeping all of Vertigo's control contact very near the top of the sphere. Lastly, the clamps were designed to concentrate most of their flexing at the bottom. This kept the upper half more rigid, ensuring a secure and consistent fit to the legs. It also gave the assembly more clearance to fit smaller spheres without rubbing on the mounts. All leg and motor mount components were again made of Delrin and machined with a CNC mill as seen in Figure 2.7.

Figure 2.7: CNC mill machining Delrin legs for Vertigo 2.1.

Drive method D utilized four motors, rather than two, which gave rise to another design decision. Purchasing two more of the original motors would be less expensive, but their shortcomings would still be present. These motors had plenty of torque, but this was accomplished by gear reductions which sacrificed speed. Vertigo would be able to balance and navigate with them, but with severely limited performance. In a fruitless attempt to rectify this problem, the gearbox was modified to eliminate

one of the reductions. This did increase their speed, but not enough. During this modification, it was found that these high-torque motors had gears made of ABS plastic, which later failed as a result of their inadequate construction. Opening up the gearbox also revealed that they were unnecessarily bulky, with cavities of unused space. Another drawback was that its method of gear reduction offset the output shaft from the center which made mounting all four in precisely the same way challenging. Lastly, the original motors required the intricate and fragile encoder assembly to measure its rotation. This was unnecessary because it had since been realized that motors and encoders could be purchased as a single unit. Faced with a long list of cogent arguments against using the original motors, it was concluded that splurging for four completely new motors was well worth the cost.

New motors for Vertigo 2.1 were carefully selected to eliminate the problems experienced with the previous set. They had to meet both the torque and speed requests of the controller, as well as have embedded quadrature encoders, concentric driveshafts, metal gearing, and be appropriately compact. To satisfy this exacting list, Faulhaber 2232V0050 gear-head DC motors with embedded quadrature encoders were eventually called upon after extensively researching the market.

Motor specifications [22]:

- Motors
  - Nominal voltage = 6 V
  - Terminal resistance < .8 Ω
  - Max. output lower = 11 W
  - Max. efficiency = 86 %

  - Max. speed = 8,000 rpm
  - Max. current = .035 A
  - Stall torque > 60 mNm
  - Permissible torque = 10 mNm
  - Speed constant = 1,200 rpm/V
  - Back EMF constant ≈ .8 mV/rpm
  - Mechanical time constant = 6 ms
  - Rotor inertia = 3.8 gcm^2
  - Max. angular acceleration 120 $10^3 rad/s^2$

- Planetary Gearhead
  - Gear ratio = 20/1
  - All metal geartrain and housing
  - Backlash ≤ 1°
  - Preloaded bearings

- Encoders
  - Liners per revolution = 512
  - Supply voltage = 5.5 VDC
  - Max. frequency = 160 kHz
  - Code disc inertia = .09 gcm^2

These outstanding motor packages offered plenty of torque and speed for this application, and were only half the volume of the 1.0 generation motor-encoder assembly.

Additionally, their stainless steel planetary gears output the driveshaft concentrically with the motor casing. At the opposite end, the embedded encoders were secured up against the motor casing. They were completely enclosed and conveniently fitted with a ribbon that not only bundled the four encoder wires, but tied in the DC motor leads as well. The ribbon terminated with a port connection that allowed an organized harness to be made that extended the leads from Qwerk so the ribbon could simply be plugged in. The location of the encoders worked perfectly with this assembly because it nestled them in the center with Qwerk where they were protected and near to their electrical connections.

It should be noted here that between the 1.0 and 2.1 designs, it was discovered that the current version of Qwerk's firmware did not support the IMU or quadrature encoders. However, the circuitry and terminals were all present, and the manufacturer was in the process of developing new firmware that promised to incorporate the seamless use of these devices. Therefore, design proceeded with these components, knowing that in the future all forms of onboard sensing would be made functional. For the time being, the responsibility of sensing would be placed on Vicon, and external motion capture system that only required small retro-reflective markers be placed on Vertigo. Detailed descriptions of the sensing methods can be found in Chapter 4.

The new actuation method also required single-row omni-wheels, allowing the originals to bemodified for use. At their new orientation, the unmodified wheels were still affective at controlling the sphere, but did not permit ground-based control. This was because the rollers were set in a cumbersome thick-walled housing that interfered with them contacting the ground. Tooling for these wheels, as with all of the cylindrical parts and modifications, was done primarily on a lathe, as seen in Figure 2.8. The objective of this modification was to maximize the exposure of the rollers. Modifications from the 1.0 generation had already milled down the material

in the roller gaps to eliminate interference with the curvature of the sphere. The new modifications faced both sides of the wheels and put chamfers at the gaps. The upper right corner of Figure 2.8 shows a side-by-side comparison of the original and modified wheels. Since the spokes that support the rollers only act in compression, and all stress is transmitted through the axels, the material above the spokes and on the walls did not serve much structural purpose. In way of torsional strength, these wheels were rated for much higher loads than those experienced in this application. Therefore, slimming down the wheels did not compromise their integrity.

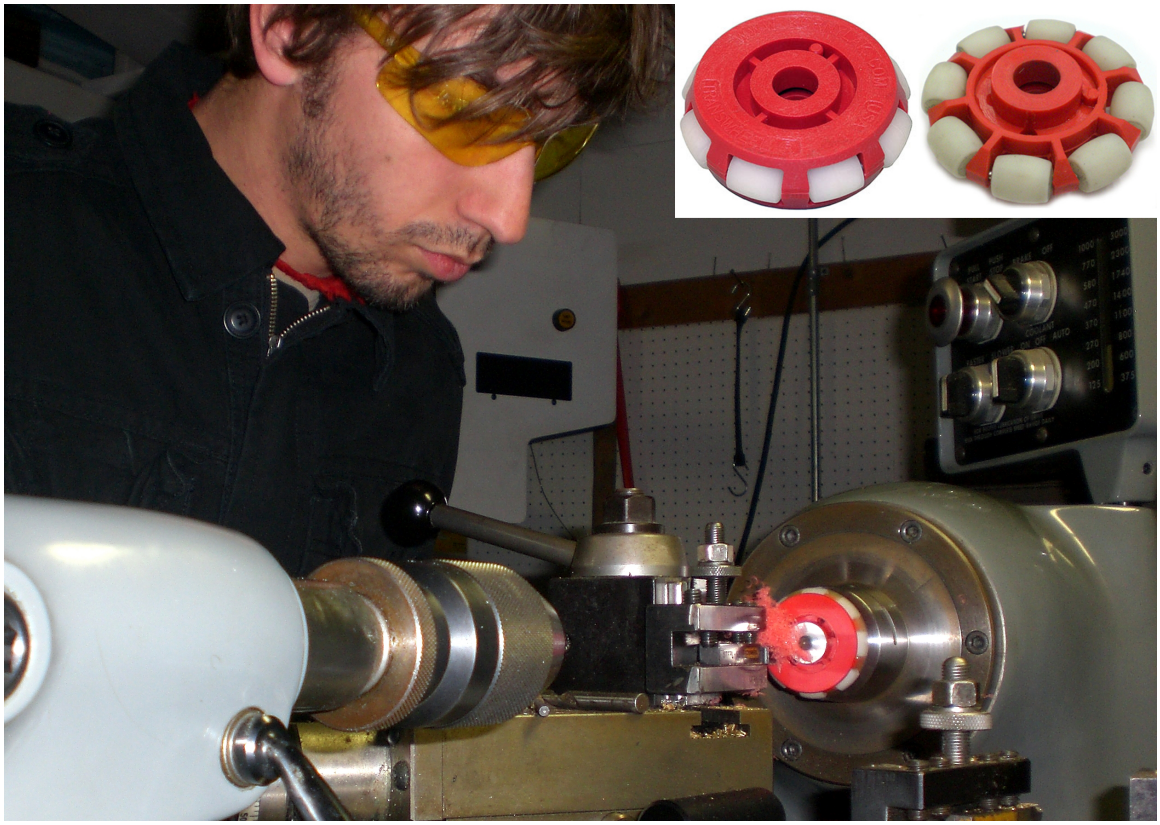


Figure 2.8: Modifying omni-wheels for Vertigo 2.1 use (photographs).

Joining these modified wheels to the new motors was the aluminum hub seen in Figure 2.9. Unlike the original hub, this did not share the burden of securing

and spacing an encoder wheel; therefore, it was made much smaller and with less complexity which required fewer machine setups to manufacture. With the success of the original fixture to draw upon, the wheels were held by a similar two-stage fixture: press fit diameter, and clamped faces. The aluminum washers were made thicker to discourage denting and bending on impacts. In the new design, the hub ends were outward facing, so rounded cap screws were used to make it easier to handle and to avoid damaging objects, floors or walls it crashed into. The hubs were secured with setscrews onto the motor driveshaft.

This design mandated that it be assembled in a specific order, starting with the hubs and setscrews, and then placing the wheels, and finally the clamping washers and end cap screws. Disassembly had to be done in the reverse order. Though it was not very complicated or time consuming, this requirement was made as a sacrifice to reduce the overall size by burying the setscrew under the wheel when assembled. To minimize play, the hub had to span as much of the driveshaft as is permitted; however, the end screw had to be threaded into the other side of the same hole, and needed its own space to thread in properly. Working out the spacing such that both components could take advantage of the same concentric hole in the hub was accomplished in both 1.0 and 2.1 design generations, but was more challenging here because there was less room to work with.

The primary goal of this design iteration was to implement the new drive mechanism that controlled all aspects of Ballbot's and Vertigo 1.0's motion, with the addition of direct yaw control. This was successfully accomplished by utilizing actuation method D. Just as importantly, this iteration was used as an opportunity to further improve upon the overall design by taking simple and well reasoned measures that had substantial impact on performance, usability and robustness. Though the new legs were made in two parts, and the motor mounts were no longer merged with the legs,

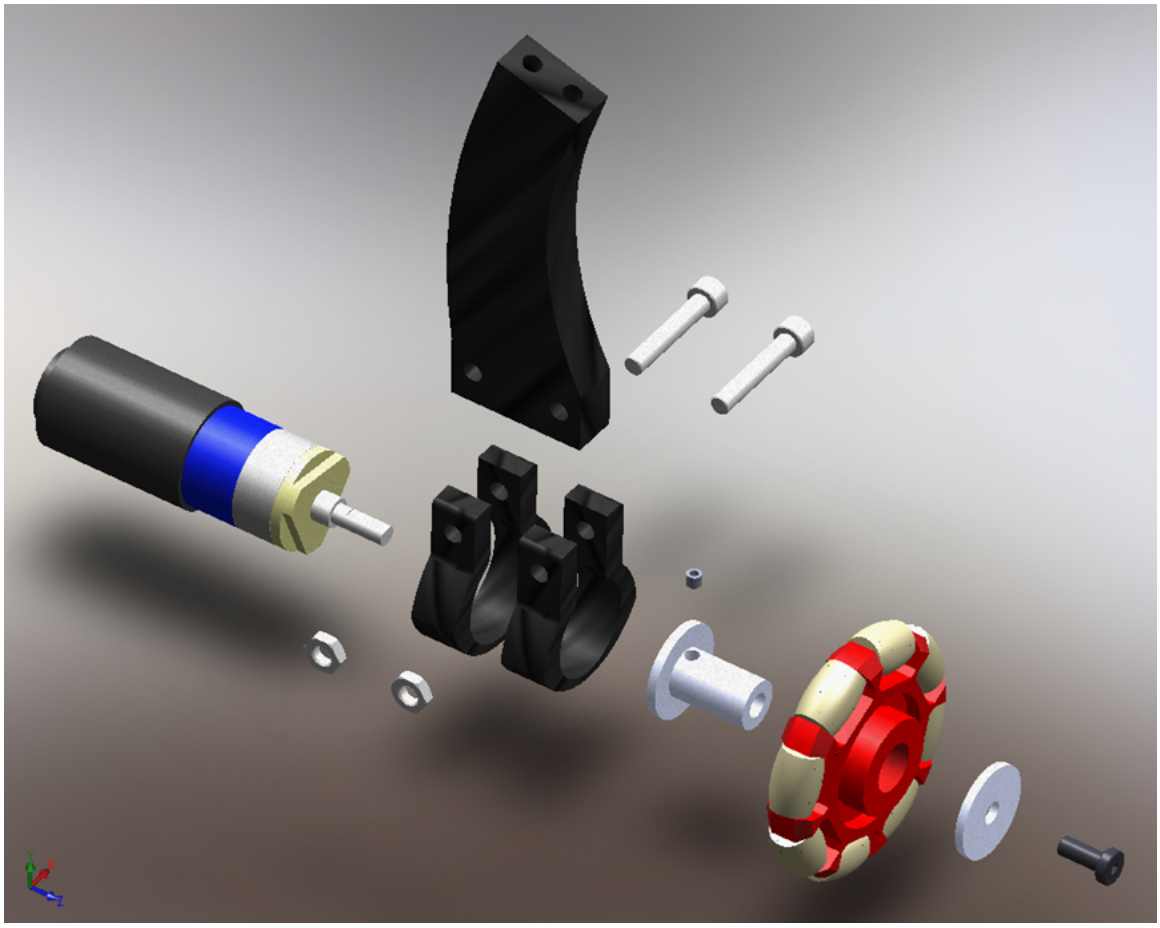


Figure 2.9: Vertigo 2.1 motor-encoder-wheel exploded subassembly (CAD).

both generations had the exact same number of unique modified and manufactured parts. This was mostly because all four limbs of the new design were identical, where the original had two with motors, and two with passive rollers. Many safety features were added or improved upon, making Vertigo 2.1 an exceptionally robust piece of hardware. The new legs offer enhanced protection for every component, including electrical connections. Wiring was made easier with larger cutouts in the legs, more clearance and better positioning of the encoder leads. The new legs were designed to not only be durable enough to survive impacts, but were sized to protect the other components.

Small features like the speaker, rounded cap screws and filleted corners made working with the robot much easier. By allowing the motors to be adjusted, even faster flexibility in the inertial and center of mass properties was achieved. At least one spare was made of every manufactured part to eliminate downtime if something broke, and two sets of batteries were purchased so that experiments would never be halted due to low power.

Aesthetics also played a role in the design, with knowledge that this may one day be marketed as a commercial platform, it will need every edge it can get for sales. Vertigo's open design balances simplicity and detail to show off its functionality without appearing austere or purpose built, even though it is. The materials, geometry and components were all chosen to form a durable system that matched its mathematical model well. In summary, the new design managed to provide solutions to Vertigo 1.0's flaws, maintain or improve upon all of its advantages, and avoid accruing new drawbacks.

2.7 Future Improvements

As work progressed with building and implementing Vertigo 2.1, even more ideas were formed that would alter or improve the hardware. In this section, these ideas are covered, and the consequences of their application discussed. Some of these concepts were devised prior to manufacturing the current design, but were omitted as their tradeoffs did not make them sensible for inclusion at this early stage.

As discussed previously in the selection of omni-wheels, this system required that only one row of rollers be used. For all existing wheels, this meant that gaps between the rollers would make the control surface discontinuous and add noise to the

measurements. The wheels used were chosen to mitigate this problem, and were successful to a degree, but the problem had not entirely vanished. This inspired a handful of new design ideas for omni-wheels, all of which would perform better than what was currently on the market, but one design stood out from the rest. The least ambitious design would simply use a larger wheel with smaller rollers to minimize the size of the gaps. This is really identical to the existing concept but with more advantageous dimensions. The most promising idea directly confronted the need for gaps. Currently, spaces exist because the spokes that support the roller axels have to fit between the rollers, pushing them apart. In addition, these gaps widen radially outward as the circumference increases, but the length of the rollers remains constant. Therefore, even designs that did not have radially facing spokes, and eliminated the inner spacing between the rollers would still have gaps at the control circumference. To overcome this, two different types of rollers could be used, with one resembling the existing rollers, and the other being larger with hollows at each end. Inside the hollows would seat the spokes and gaps for the smaller rollers. The outer surface of both types of rollers would be turned to the same radius as the assembled wheels. This configuration allows the two types of rollers to be placed as close as tolerances allow, forming an almost perfectly continuous surface.

Inclusion of these wheels would greatly decrease the demands on the estimator because measurement noise from the drive mechanism would be nearly eliminated, and prescribed control would more accurately be carried out, fitting the mathematical model more accurately and improving predictions. The drawback to this design is that they are extremely intricate and complex. To make just one wheel that had 6 of each type of roller, 28 tiny parts would have to be designed and made from scratch; that is about twice the number it takes to manufacture the whole of Vertigo 2.1. To make four of these would be quite time consuming, but in the future, their benefit

may be deemed worth the effort.

To this point, the 2.1 legs have only solved problems, not caused any; however, this is not to say that problems cannot arise in the future as trajectories increase in complexity, and the dynamics of the system are tested to their limit. The number of legs is an example of this, and the decision to include four rather than three was nontrivial. Three legs are all that is needed to support and control the robot on top of the sphere with this actuation method, allowing for minimal weight and number of parts. It would also make the system equal-actuated, with three motors to control three degrees of freedom. However, there was a fundamental concern with only having three vertical legs in our orthogonal universe: the configuration is limited to being asymmetrical at best. Therefore, even if the actuators are applying orthogonal control, the product of inertia, a consequence of omitting the symmetry of four legs, would still couple their effect on the system. Accordingly, using only three legs would not fit the planar and decoupled mathematical model, and controller design would be more challenging. Another advantage to using three legs is that it would ensure that all control contact be firmly secured to the surface of the sphere, increasing the frictional force and reducing the likelihood of slippage. In the four legged design, this issue was mitigated by holding tight tolerances when manufacturing the uniform length legs, and designing the legs to flex slightly. This permitted flexing has not yet been problematic, but may be in the future if more extreme control induces undesired vibrations. If this does occur, the addition of simple struts that brace the lower legs would restrict their motion and prevent oscillation.

Another possible issue is with the battery orientation. The battery pack is currently composed of two identical bricks that are bundled together to make removal and installation convenient. Their diagonal orientation causes a dynamically coupling product of inertia that has not yet been addressed, but may need to be in the future.

The solution to this is fast and simple. Simply mounting them perpendicularly will cancel out their products of inertia and improve the system's dynamics.

Vertigo's design gives it a high degree of flexibility in its center mass and inertia. However, this reconfiguration requires some disassembly which may be too time consuming for a given experiment, depending on what is being moved. To make the physical properties of Vertigo even more flexible, simple appendages with sliding and interchangeable weights could be added. They could be designed to mount in several directions and support weights of various magnitudes in different locations. This would help experimentally prove the theoretically suggested concept that it is easier to balance with a higher center of gravity. It would also help explore the affects that coupling inertias have on the system dynamics.

Lastly, Vertigo is currently initialized in Vicon by being placed on the ground as a reference. The controller is then applied, and then the body placed on the sphere manually. This should be done quickly and accurately to prevent human interference with the control, almost to the point where it is aligned a short distance above the sphere, and dropped. If hands are trying to guide the body onto the sphere while the controller is also trying to adjust its position, the system oscillates and can even diverge because both inputs are acting in the same direction, causing overshoot. Inclusion of retractable, statically stabilizing legs may be a solution to avoid this. A simple design where servomotors retract the leg extensions as soon as balancing control is applied, and quickly extends them again when control is terminated would eliminate the need for placing the body on the sphere, or making sure it is off before ending control. It would also allow Vicon to initialize the system at its proper balancing height, and no bias would have to be included to account for the added height from the sphere. The Ballbot team has implemented a similar set of retractable legs with great success.

Chapter 3

Mathematical Modeling and Analysis

3.1 Introduction

The intent of this chapter is to derive the equations of motion for Vertigo and conduct an extensive controllability and observability analysis on these equations. A trustworthy mathematical model is invaluable when designing reliable and affective estimators and controllers. In development, accuracy and convolution must be balanced using reasonable assumptions to yield dependable equations that avoid excess computational cost. Analysis of these equations is also important; it helps to verify their accuracy and establish the objective of the control and estimation processes that will be applied later on. This chapter derives the nonlinear equations of motion for sphere-based Vertigo, and then shows how they are linearized for use in linear estimation and control theory. Ground-based equations of motion are also derived, and will be used for control in that configuration. Observability and controllability analyses are then conducted for the states and design parameters of the sphere-based

equations of motion. This development should give a quality model that is well understood and can reliably be used in the remainder of this thesis.

3.2 Mathematical Model

3.2.1 Introduction

Having a mathematical model of a system is required for most analysis, control and estimation methods. In general, there are two approaches for obtaining a model: system identification, and mathematical derivation. System identification is a broad topic that is composed of a diverse collection of methods and algorithms; in general they are all designed to extract dynamical models from measurement data. System identification is most advantageous when applied to complex or flexible systems. This is because it does not require knowledge of the system that is to be modeled. Deriving equations of motion mathematically does require knowledge of the system, so it is most readily done for systems with well known dynamics, such as rigid bodies. Derivation does not require previously collected data, so it can be done for theoretical systems or in the preliminary stages of development, before data is available. Having the equations before the system is even built means that they can be used in simulation as proof of concept. Additionally, their analysis can help detect unforeseen problems that may be addressed early on or maybe avoided all together with design modifications.

Vertigo's model was derived mathematically because rigid body dynamics can be assumed, and data was not available when first modeling the system. The equations

of motion were developed as Vertigo was being designed, and they were analyzed to help make design decisions. System identification methods can also be used to verify and improve the model after implementation. Having an existing model will aid in that process by focusing efforts appropriately.

In the derivation that follows, a planar model is assumed. This is a simplification of the actual system, but a reasonable one because Vertigo was intentionally designed to be highly symmetrical so the motion will not only be the same in both the sagittal and coronal body planes, but the product of inertia will not couple their motion significantly. Using a simplified model helped accelerate the project by requiring management of simpler equations, and turned out to be a beneficial decision. In the planar model yaw is not incorporated; however this is not debilitating because control superposition can be applied in moderation for trajectories that do not demand extravagant yaw control (see Control Superposition section in Appendix A). Once the project is further along and more complex trajectories are attempted, the more inclusive dynamics can be exploited. Since Ballbot does not have control in yaw, Carnegie Mellon's team was also able to derive its equations of motion for a planar model [3]. Their model will be followed here because it is very well reasoned with logical assumptions, and it closely describes Vertigo's planar motion.

A vast majority of introductory analysis, estimation and control techniques are designed for use on linear systems. Nonlinear systems, such as Vertigo, can take advantage of these methods with local accuracy if the system is linearized about the point of interest. Vertigo's nonlinear equations of motion will be linearized with a first order Taylor series expansion approach.

Ground-based equations of motion are then developed for use in that configuration. The model used yields linear equations, so linearization is not necessary. All three sets of equations proved to be sufficiently accurate for drawing conclusions

about the characteristics of Vertigo's motion, and were used for the analysis, estimation and control that follows. The reconfigurable nature of this platform means that the parameter values may change slightly between the various simulations and implementations, but the same model structures are used throughout.

Mathematical models will have errors based on measurement uncertainties and simplifying assumptions. Examples of these include the inertias, friction coefficients and the relation between signal input and actual torque that is applied by the motors. To minimize the impact of these errors, filters can be applied that account for uncertainty, or even predict unknown parameters.

3.2.2 Model Derivation

Viewed by many as the most important step in control and simulation, modeling is charged with balancing accuracy and practicality. Following the Ballbot team's procedure, Lagrangian mechanics are used here to develop Vertigo's nonlinear equations of motion. This method starts with the identification of kinetic energy $K = K_b + K_B$ and potential energy $V = V_b + V_B$, where the subscript b represents the sphere, and B represents the body of the robot. From the energies, the Lagrangian $\mathcal{L}$ can be constructed and the Euler-Lagrange equations of motion calculated as follows.

$$\mathcal{L}(\mathbf{q}, \dot{\mathbf{q}}) = K - V \tag{3.1}$$

$$\frac{d}{dt}\mathcal{L}_{\dot{q}} - \mathcal{L}_q = T \tag{3.2}$$

Where T is the input torque for the motors and $\mathbf{q}$ is the state vector.

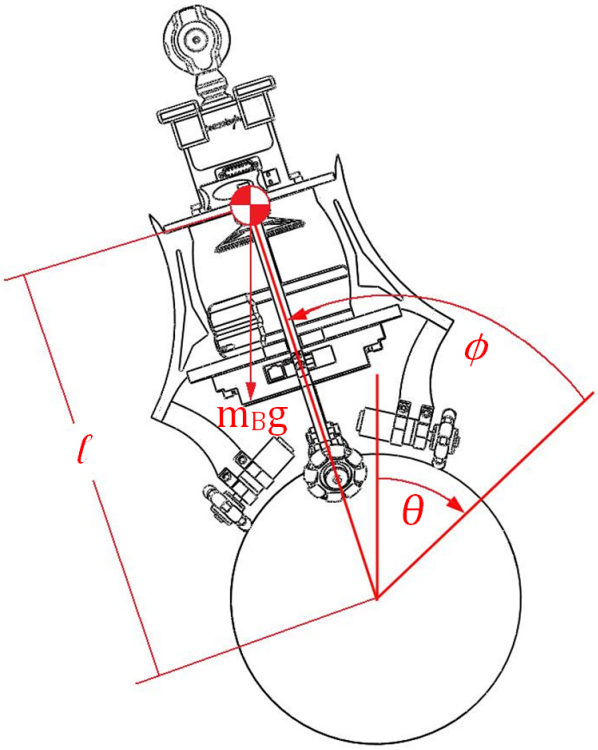


Figure 3.1: Planar sphere-based Vertigo model.

Figure 3.1 depicts the state and parameter convention that will be used throughout this thesis for the sphere-based configuration. It is worth noting here that the angle of the body ϕ is measured with respect to the angle of the sphere θ. This is done for mathematical convenience; however, it makes some of the results and simulations less intuitive to interoperate. For this reason many of the results include representations that are easier to visualize in addition to the state results.

The following assumptions are made during this formulation:

1. Rigid sphere and body
2. No slip between ball and ground
3. No slip between ball and actuation

4. Inputs are torques applied between roller and ball
5. Sagittal and coronal plane are symmetric and decoupled
6. Neglect friction in actuator rollers
7. Neglect dynamic friction
8. Origin is at the center of the sphere
9. Yaw coupling is negligible

The first step in the Lagrangian formulation is to define the kinetic and potential energy for the body and sphere.

Sphere:

$$K_b = \frac{I_b\dot{\theta}^2}{2} + \frac{m_b\left(r_b\dot{\theta}\right)^2}{2} \tag{3.3}$$

$$V_b = 0 \tag{3.4}$$

Body:

$$K_B = \frac{m_B}{2}\left(r_b^2\dot{\theta}^2 + 2r_b\ell\left(\dot{\theta}^2 + \dot{\theta}\dot{\phi}\right)\cos\left(\theta+\phi\right) \ + \ell^2\left(\dot{\theta}+\dot{\phi}\right) + \frac{I_B}{2}\left(\dot{\theta}+\dot{\phi}\right)^2\right) \tag{3.5}$$

$$V_B = m_B g\ell\cos\left(\theta+\phi\right) \tag{3.6}$$

Where, m_b and m_B are the mass of the sphere and body respectively, I_b and I_B are the inertias of the sphere and body respectively, r_b is the radius of the sphere, and ℓ the length from the body's center of mass to the center of the sphere. The total energies are considered for the derivation, and are calculated as follows. Total kinetic

energy is $K = K_b + K_B$, and total potential energy is $V = V_b + V_B$. Defining the system configuration vector as $\mathbf{q} = [\theta \quad \phi]^T$, the Lagrangian formulation is continued with,

$$\mathcal{L}(\mathbf{q}, \dot{\mathbf{q}}) = K - V \tag{3.7}$$

The input torque that is applied from the motors is defined by τ. The viscous friction is modeled by defining

$$D(\dot{\mathbf{q}}) = \begin{bmatrix} \mu_\theta \dot{\theta} \\ \mu_\phi \dot{\phi} \end{bmatrix} \tag{3.8}$$

where μ_θ is the viscous damping coefficient for friction between the sphere and ground, and μ_ϕ is the coefficient for friction between the sphere and body. The Euler-Lagrange equations can then be expressed with this notation as

$$\frac{d}{dt}\mathcal{L}_{\dot{\mathbf{q}}} - \mathcal{L}_{\mathbf{q}} = \begin{bmatrix} 0 \\ \tau \end{bmatrix} - D(\dot{\mathbf{q}}) \tag{3.9}$$

The derivative of this gives the equations of motion, which are rearranged to take the form

$$M(\mathbf{q})\ddot{\mathbf{q}} + C(\mathbf{q}, \dot{\mathbf{q}}) + G(\mathbf{q}) + D(\dot{\mathbf{q}}) = \begin{bmatrix} 0 \\ \tau \end{bmatrix} \tag{3.10}$$

Where,

$$M(\mathbf{q}) = \begin{bmatrix} \Gamma_1 + 2m_B r_b \ell cos(\theta + \phi) & \Gamma_2 + m_B r_b \ell cos(\theta + \phi) \\ \Gamma_2 + m_B r_b \ell cos(\theta + \phi) & \Gamma_2 \end{bmatrix} \tag{3.11}$$

is the mass and inertia matrix with,

$$\Gamma_1 = I_b + I_B + (m_b + m_B)\, r_b^2 + m_B \ell^2 \tag{3.12}$$

$$\Gamma_2 = I_B + m_B \ell^2 \tag{3.13}$$

The vector of Coriolis and centrifugal forces is defined by

$$C(\mathbf{q}, \dot{\mathbf{q}}) = \begin{bmatrix} -m_B r_b \ell sin(\theta + \phi) \left(\dot{\theta} + \dot{\phi}\right)^2 \\ 0 \end{bmatrix} \tag{3.14}$$

and the vector of gravitational force components is

$$G(\mathbf{q}) = \begin{bmatrix} -m_B g \ell sin(\theta + \phi) \\ -m_B g \ell sin(\theta + \phi) \end{bmatrix} \tag{3.15}$$

By defining $u = \tau$, and $\mathbf{x}$= $\begin{bmatrix} \mathbf{q} & \dot{\mathbf{q}} \end{bmatrix}^T$, the equations of motion can be expressed in nonlinear state space form as,

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{\mathbf{q}} \\ M(\mathbf{q})^{-1} \left(\begin{bmatrix} 0 \\ u \end{bmatrix} - C(\mathbf{q}, \dot{\mathbf{q}}) - G(\mathbf{q}) - D(\dot{\mathbf{q}}) \right) \end{bmatrix} \triangleq f(\mathbf{x}, \mathbf{u}) \tag{3.16}$$

3.2.3 Linearization

Many control, estimation and analysis methods are for linear systems, so Vertigo's nonlinear equations must first be linearized. The linearization used here is based on keeping only the linear terms of a Taylor series expansion about an operating point. This is often referred to as Jacobian linearization because, for a state space

formulation, $\dot{\mathbf{x}} = A\mathbf{x}+B\mathbf{u}$, this can be reduced to taking the Jacobian of the nonlinear equation with respect to the states, to obtain the A matrix, and with respect to the inputs for the B matrix. This process takes on the form,

$$A = \left.\frac{\partial \dot{\mathbf{x}}}{\partial \mathbf{x}}\right|_{\substack{\mathbf{x}=\overline{\mathbf{x}} \\ \mathbf{u}=\overline{\mathbf{u}}}} \quad B = \left.\frac{\partial \dot{\mathbf{x}}}{\partial \mathbf{u}}\right|_{\substack{\mathbf{x}=\overline{\mathbf{x}} \\ \mathbf{u}=\overline{\mathbf{u}}}} \quad C = \left.\frac{\partial \mathbf{y}}{\partial \mathbf{x}}\right|_{\substack{\mathbf{x}=\overline{\mathbf{x}} \\ \mathbf{u}=\overline{\mathbf{u}}}} \quad D = \left.\frac{\partial \mathbf{y}}{\partial \mathbf{u}}\right|_{\substack{\mathbf{x}=\overline{\mathbf{x}} \\ \mathbf{u}=\overline{\mathbf{u}}}} \tag{3.17}$$

where the barred values represent the point being linearized about. When the system linearization is transformed to the origin, all states are zero, and the state space becomes,

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -43.83 & -43.83 & -.0342 & -.0878 \\ 79.31 & 79.31 & .0439 & -.1228 \end{bmatrix} \quad B = \begin{bmatrix} 0 \\ 0 \\ -43.9 \\ 61.4 \end{bmatrix} \tag{3.18}$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad D = [\mathbf{0}] \tag{3.19}$$

This model represents Vertigo with its completed communications architecture, where all onboard sensing is assumed fully operational. Accordingly, C is the identity matrix because all states are measured. The extreme size of the symbolic state space equations made them unreasonable for inclusion here, so the above model with numerical values is given. Note that these values represent the parameters for a spe-

cific configuration of Vertigo, and is linearized about the vertical equilibrium. As the equation parameters and the point of linearization changes, so too will these values, but the general results that this model yields are still valid.

3.2.4 Ground-Based Derivation

As mentioned before, Vertigo's actuator orientation and axial symmetry means that the full state space equation with x and y position and velocity is decoupled and identically redundant for both directions. Therefore, control and implementation can be carried out for a planar model, and applied to both planes independently. The continuous equations of motion for the ground-based robot were put in standard state space form $\dot{\mathbf{q}} = A\mathbf{q} + Bu$, with the output taking the form $y = C\mathbf{q} + Du$. With the state vector defined as $\mathbf{q} = \begin{bmatrix} x & \dot{x} \end{bmatrix}^T$ and control as $u = T$, the model can be expressed as,

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{2}{mr} \end{bmatrix} T \tag{3.20}$$

$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0 \end{bmatrix} T \tag{3.21}$$

Where r, m and T are the radius of the wheels, mass of the robot and imposed torque of an individual motor, respectively. This model makes the assumptions that all components are rigid bodies, dynamic affects can be neglected, and opposing

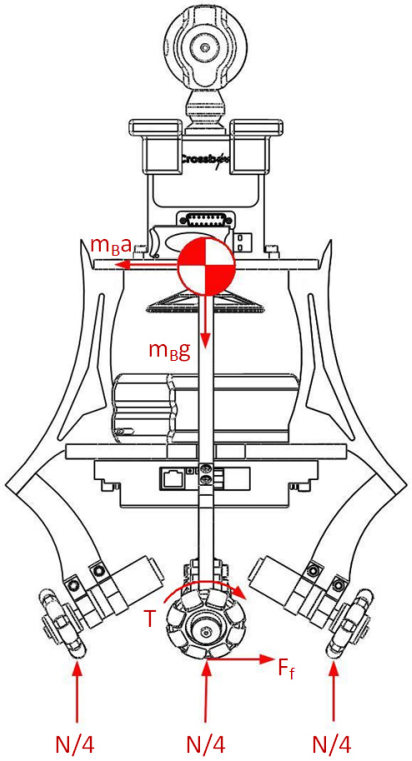


Figure 3.2: Ground-based free body diagram.

frictions at the wheels are negligible. As a result of the dynamic simplifications, all points of contact with the ground are assumed to carry equal portions of the total normal force N. In reality, this is not true for two main reasons: all four wheels will not always be in contact with the ground due to uneven terrain and the discontinuity of the wheels, and as the body accelerates, the leading wheel experiences less normal force than the trailing wheel (discussed further in the Implementation Challenges section). Neglecting opposing frictional forces mainly assumes that the rollers on the wheels do not have any appreciable resistance since they are designed to allow slip. If it were deemed significant, modeling this friction would depend on the system's dynamics and estimated coefficients, making it highly inaccurate.

During some of the ground-based controller development, discrete time equations

of motion (with $T_s = 0.04$) were used.

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} 1 & 0.04 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0.0240 \\ 1.1976 \end{bmatrix} T \tag{3.22}$$

$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0 \end{bmatrix} T \tag{3.23}$$

The sampling time was chosen based on the average sampling time of the implemented closed loop (see the Implementation Challenges section for more details). The discretized equations were obtained with MATLAB's c2d command.

3.3 Analysis

3.3.1 Introduction

The objective of this section is to determine the conditions for which sphere-based Vertigo can be controlled, and whether its states can be reliably determined if provided knowledge of its inputs and outputs. Investigating these characteristics will help build an understanding of the system and establish what should be expected of the control and estimation processes that it will rely on for balance and navigation. To accomplish this, an extensive controllability and observability analysis is conducted on the linearized model. It is important to note that a linearized model may well capture

the dynamics of a system near its equilibrium, but subject to large perturbations, the inherent nonlinearities will decrease the validity of the simplification.

In this analysis, the observability $\mathcal{O}$ and controllability $\mathcal{C}$ matrices are calculated respectively as,

$$\mathcal{O} = \begin{bmatrix} \mathbf{C} & \mathbf{C}A & \cdots \mathbf{C}\mathbf{A}^{n-1} \end{bmatrix}^T \tag{3.24}$$

$$\mathcal{C} = \begin{bmatrix} \mathbf{B} & \mathbf{A}B \cdots \mathbf{A}^{n-1}\mathbf{B} \end{bmatrix} \tag{3.25}$$

where n is the number of states. The system is observable if the observability matrix has full column rank, and controllable if the controllability matrix has full row rank. All four states will be varied over a range of values, and this calculation carried out for the relinearized system at each point. The rank criterion for $\mathcal{O}$ and $\mathcal{C}$ only gives insight as to whether the system is, or is not observable or controllable, it tells nothing of a measure or degree of observability and controllability. To dig deeper into this, the inverse of the condition number and determinants of these matrices are examined.

Many factors contributed to the structural development of this platform. Balancing machines are fundamentally dependant on the location of their center of gravity (length of the inverted pendulum), so understanding the nature of this dependence was important to the design process. Furthermore, Vertigo was intentionally created to be reconfigurable, which allows its center of mass to be altered; so it was necessary to determine how the physical changes should be expected to affect the performance, making experiment design more predictable. Two main components assemble to comprise this system, the sphere, and the body (Vertigo). To explore the sensitivity to these components, a controllability and observability analysis is done for both while varying their dimensional and mass parameters. As before, the determinant and condition number are also considered for further evaluation.

3.3.2 State Observability and Controllability Analysis

For a system to be controllable, an input must exist that will transfer any state to any other state in a finite amount of time. It assumes that the linear system has infinite control magnitude available, and that any trajectory may be taken. This is an extremely important preliminary calculation for system design to ensure that it is possible for the objectives to be met. The controllability is determined by checking the rank of the controllability matrix $\mathcal{C}$.

$$rank(\mathcal{C}) = \ rank(\begin{bmatrix}\mathbf{B} & \mathbf{A}B \cdots \mathbf{A}^{n-1}\mathbf{B}\end{bmatrix}) \tag{3.26}$$

If the controllability matrix has full row rank, then the system is controllable. In this case there are four states, so the controllability matrix must have rank four.

If a system is observable, it means that for any initial state there exists a finite, positive time where a record of the input and output is enough to uniquely determine that initial state. To determine observability, the observability matrix $\mathcal{O}$ must be found to have full column rank.

$$rank(\mathcal{O}) = rank(\begin{bmatrix}\mathbf{C} & \mathbf{C}A & \cdots \mathbf{C}\mathbf{A}^{n-1}\end{bmatrix}^T) \tag{3.27}$$

Just as in controllability, for this system, full rank is four.

In MATLAB, the controllability and observability were calculated while varying all four states, and the rank of the matrices at each point were stored in four-dimensional matrices. Figures 3.3 and 3.4 clearly show that this system has full rank for all values of theta and phi under consideration. This range of values was logically chosen based

on the physical system's characteristics. A range of ±1 radian is nearly ±60°, which tests a comprehensive span of possibilities because by this point Vertigo would have fallen off of the sphere it rests on.

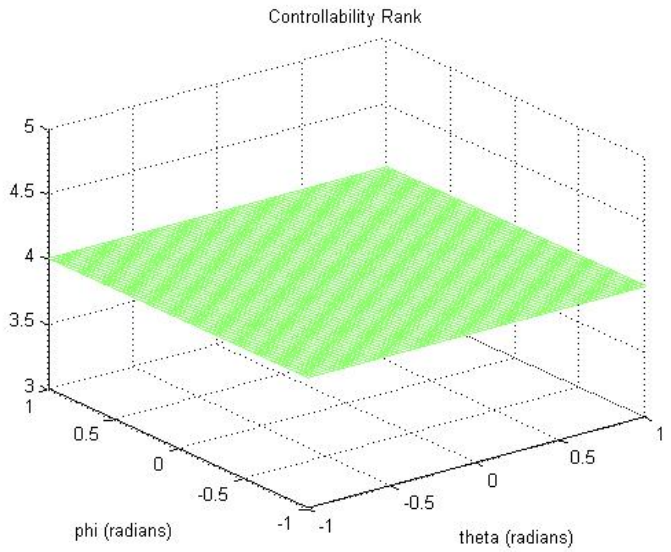


Figure 3.3: Rank of controllability matrix while varying θ and ϕ.

Many of the results in this thesis are presented in terms of $\beta = \theta + \phi$, the angle of Vertigo with respect to vertical, because it is more intuitive, and combines the four states into just two values. However, this analysis does not lend itself to such a representation because its results are not unique. For given rates, multiple combinations of angle states may give the same β, but combinations do not necessarily share controllability and observability characteristics.

The rank values in this analysis only inform of whether or not the system is controllable or observable. To further look into what is taking place as these variables change, the inverse of the condition number was calculated and plotted for both matrices. The condition number of a matrix is the largest singular value divided by the smallest singular value. The singular values of a matrix $A_{(mXn)}$, where m = n,

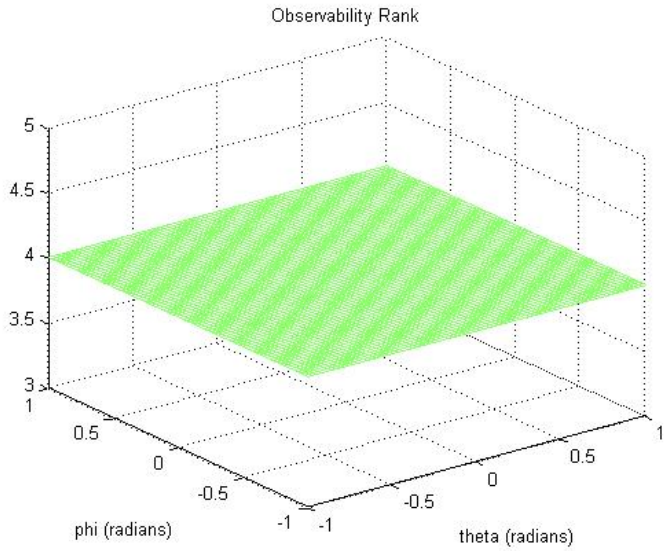


Figure 3.4: Rank of observability matrix while varying θ and ϕ.

are the positive roots of the eigenvalues of $A^H A$. Matrix A is non-singular if and only if (iff) all of its singular values are greater than zero. It can be shown that if $\mathcal{C}$ can be put in to Control Canonical Form (CCF), it is completely controllable; in order to do so, it must be nonsingular because its inverse must be taken. This is also true of $\mathcal{O}$ for the Observer Canonical Form (OCF). Therefore, the condition number provides a sensitivity bound for the solution of linear equations of the matrix. By definition, the condition number is always greater than or equal to one, so calculating the inverse of the condition number gives an easy to visualize scaled measure between one and zero. If the inverse of the condition number is close to one, the matrix is said to be "well conditioned." This means that the inverse can be computed with high accuracy. A smaller inverse condition number means that the matrix is nearing singularity, and computation of its inverse, or the solution to linear systems of equations is prone to large numerical errors. If the inverse of the condition number is exactly zero, that corresponds to a condition number of infinity, and the matrix is singular. Here,

this would translate to not being controllable or observable for the controllability or observability matrices respectively.

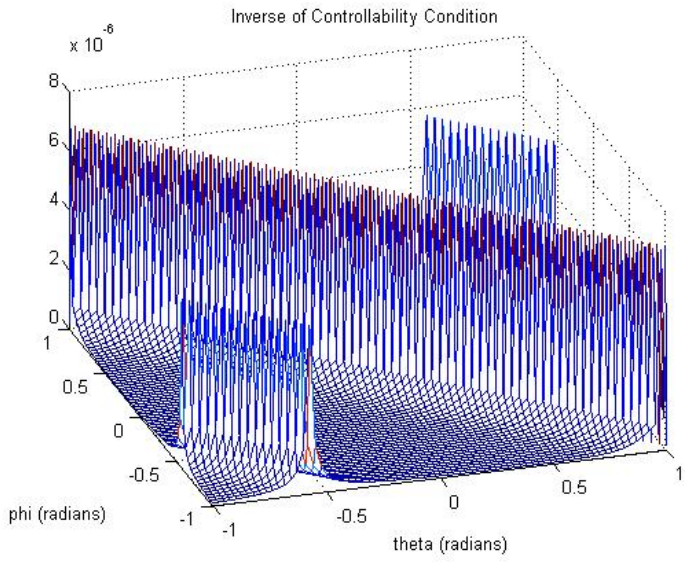


Figure 3.5: Inverse condition number of controllability matrix while varying θ and ϕ.

From Figures 3.5 and 3.6, it can be seen that the controllability and observability characteristics are definitely affected by the state angles, an observation that could not be made from the controllability and observability pots. For controllability, in Figure 3.5, consider the profile curve for fixed ϕ at zero, and varying θ. θ is defined with respect to ϕ, so fixing ϕ at zero means that the body and sphere will rotate together. Therefore, it makes sense that there is a peak about the unstable equilibrium point, because it would take less control input to move to another state. It is also easy to understand that if the body is initially tipped at an angle, more control would be required to work against gravity and resume balance. The rotation of the sphere has nothing to do with the gravitational component of the system or controllability, except for the fact that the angle of the body is defined with respect to it. This is why the general shape of the profile remains the same along curves of fixed θ, and only

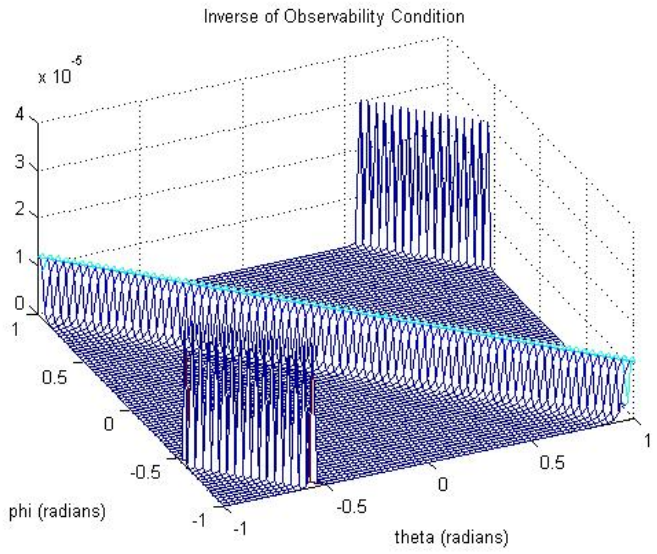


Figure 3.6: Inverse condition number of observability matrix while varying θ and ϕ.

shifts to the right or left, giving the symmetrical nature of these plots. In Figure 3.6, if the constant $\phi = 0$ profile is considered again, the results show that the system is more easily observed near the equilibrium. This suggests that, given the input and output history, it is easier to determine the initial condition when it started closer to vertical. Again, the angle definition is the reason for the symmetrical shape of the plot in Figure 3.6. For both plots, the scaling shows that these peaks are small, so the state variations do not have an overwhelming impact.

As explained before, the inverse of the controllability matrix must exist to be put in to CCF; therefore, the determinant must be nonzero. To further illustrate this, the determinant of the controllability matrix was calculated and plotted for analysis as well. Figure 3.7 shows how these results agree with the findings of the inverse condition number. This analysis was not done for observability because it does not always take the form of a square matrix, and its determinant is not always calculable; however, techniques exist that account for this.

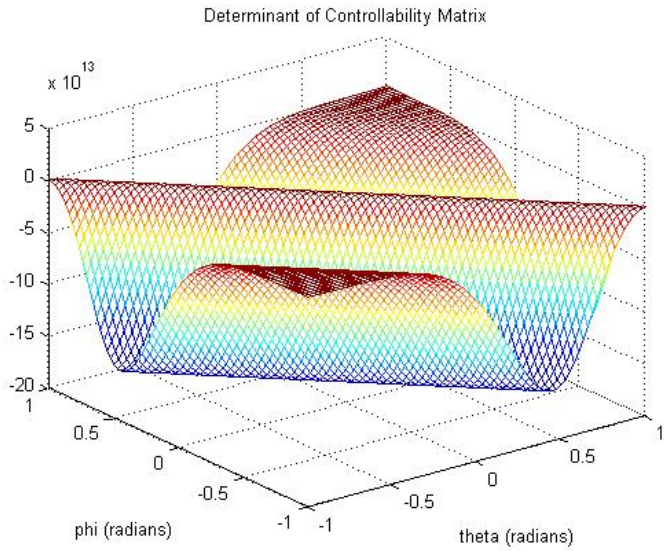


Figure 3.7: Determinant of controllability matrix while varying θ and ϕ.

To this point, the controllability and observability analysis has been devoted to the θ and ϕ state combination as they are varied. The same analysis was done for all six combinations of states pairs as the four states were varied. Figures 3.8 and 3.9 show that as the two rate states $\dot{\theta}$ and $\dot{\phi}$ were varied, the controllability and observability are still preserved.

When this rate analysis was extended to the inverse condition numbers, it was found that for both observability and controllability, the value was a constant for all rates. Accordingly, the mesh plot for the inverse condition number of the controllability matrix was a flat plane at 1.67×10^{-7}, and the inverse condition number of the observability matrix was a flat plane at 1.17×10^{-6}. The plot of the determinant of the controllability matrix was also a constant value at 1.6×10^{11}. Constant values mean that these states do not affect the controllability or observability. This was confirmed upon closer inspection of the A matrix, where it can be shown that these values cancel each other out, and only the angle states can change the A matrix.

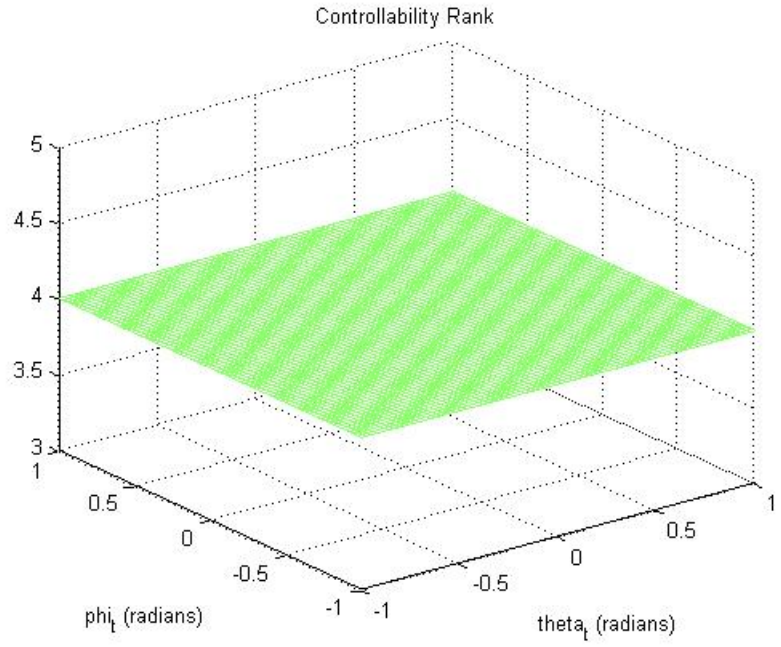


Figure 3.8: Rank of controllability matrix while varying $\dot{\theta}$ and $\dot{\phi}$.

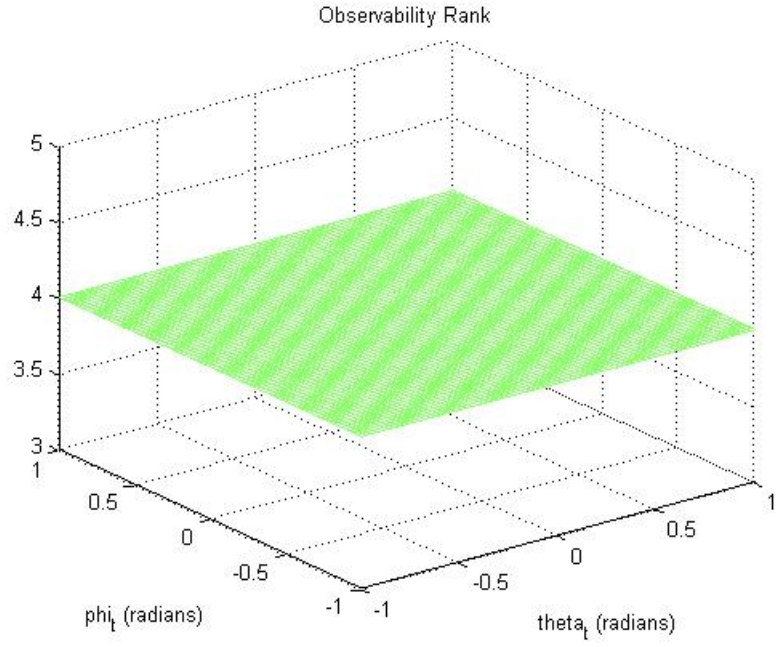


Figure 3.9: Rank of observability matrix while varying $\dot{\theta}$ and $\dot{\phi}$.

Therefore, all of the state varying characteristics were captured in the angle state analysis. As a result of the controllability and observability being invariant to rate states, the four remaining state combination analyses yielded no new results, and were not included for this reason.

3.3.3 Design Parameter Observability and Controllability Analysis

The physical parameters of a system can greatly affect the mathematical modeling. Vertigo is a testbed for dynamics and controls experiments, and for that reason was designed to be reconfigurable, permitting changes to the center of mass location and magnitude. In addition, new design iterations are always underway, so it is important to determine how these physical changes in the system affect its performance. The two main components of Vertigo's assembly are the sphere, and the body. To explore the sensitivity to these components, a controllability and observability analysis was done for both constituents, while varying their dimensional and mass parameters. This analysis also looks into the determinant and inverse condition number for further evaluation. Performing this analysis sheds light on how the equations of motion are affected by uncertainty in these parameters, which will help in the estimation and filtering to come.

Vertigo Design Analysis

First, controllability and observability are checked by plotting the rank of these two matrices while the design parameters, body mass of Vertigo and length from the

center of the sphere to the center of mass of Vertigo ℓ, are varied.

Figures 3.10 and 3.11 show that the system is both controllable and observable for all parameter variations because the rank of both matrices is four at each point, matching the number of states. Now the inverse condition number of these two matrices is considered to evaluate the parameter's affect on how well conditioned they are. The contours in Figures 3.12 and 3.13 show the inverse condition number as the parameters are varied. The vertical blue lines mark the parameters to which Vertigo is currently configured.

Figure 3.12 reveals that as the length from the center of the sphere to the center mass of the body increases, the controllability matrix monotonically becomes more ill conditioned. To help understand this, consider the following scenario: Vertigo is required to return to a vertical stature from an initial angle of 10 degrees. This plot shows that more control effort would be required for the system to recover if the length is greater. This makes sense because, for a fixed angle, the horizontal lever arm creating a torque with gravity grows with this parameter. However, if this was the only affect of increasing this length, the plots would decrease at a constant slope because of the linear relationphip; this is clearly not the case. The reason is that the rotational inertia also increases as ℓ increases, causing the body to fall more slowly. Additionally, the lever arm from the control to the center of gravity also increases. Both of these tend to decrease the amount of control effort required to recover. This explains the concave characteristics with increasing length. It is important to note that this fixed angle example assumes the same initial angle of departure, but in practice larger rotational inertia cases take longer to fall, and will not experience as much departure in a given period of time. So, when time is included, a longer ℓ is actually perceived as "easier" to control.

For the same plot, if we inspect how the condition changes as Vertigo's mass

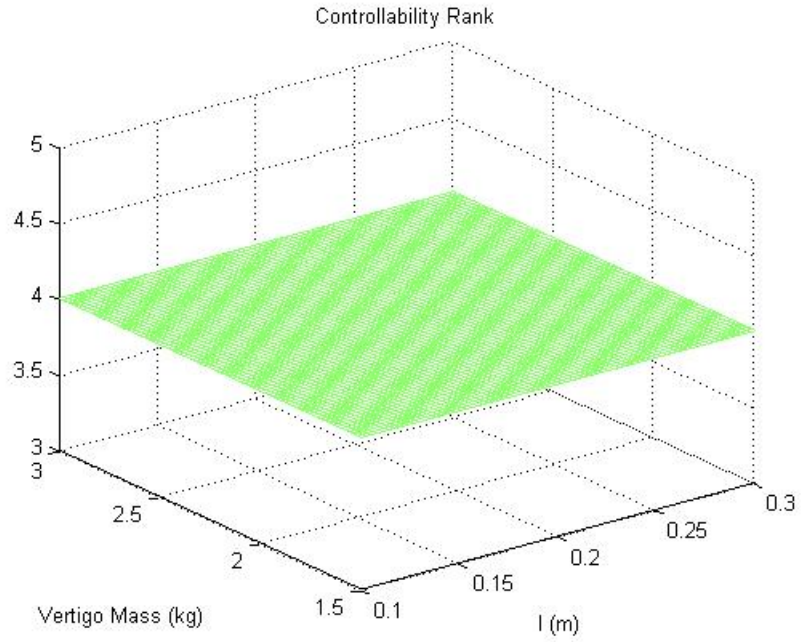


Figure 3.10: Rank of controllability matrix while varying $\dot{\theta}$ and $\dot{\phi}$.

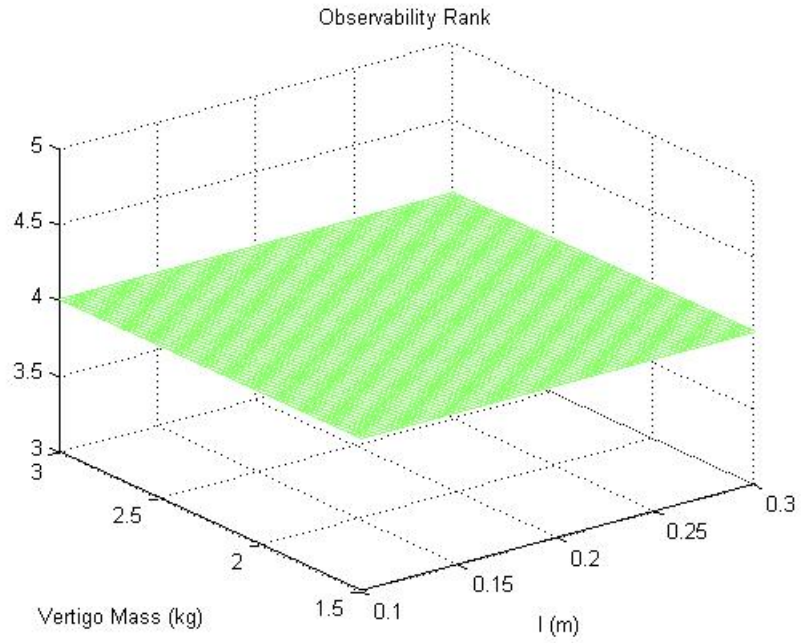


Figure 3.11: Rank of observability matrix while varying $\dot{\theta}$ and $\dot{\phi}$.

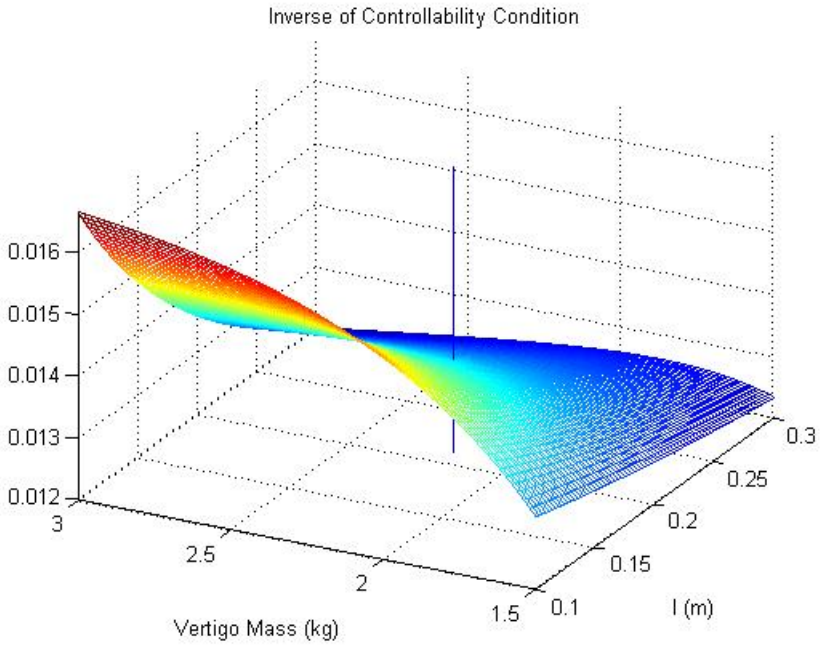


Figure 3.12: Inverse condition number of controllability matrix while varying Vertigo mass and ℓ.

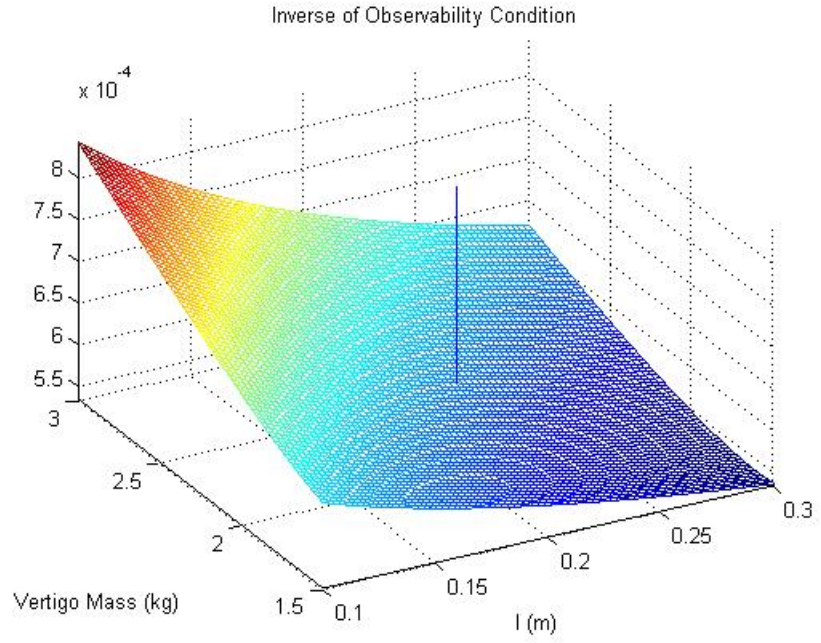


Figure 3.13: Inverse condition number of observability matrix while varying Vertigo mass and ℓ.

increases, a definite ridge can be seen where the matrix is best conditioned. Consider again the initial angle recovery scenario. Increasing mass will increase the moment with gravity, making recovery more difficult. However, linear inertia also increases, so it has greater resistance to motion, making it easier to move the sphere underneath it and recover from the initial angle. The increased mass is assumed to be at a point, so the rotational inertia about the center of mass does not change, and the control torque required to rotate it does not change. However, the rotational inertia with respect to the center of the sphere does increase, and the body will fall more slowly. The slower falling body, with no additional control requirements, acts to improve controllability with increasing mass. These phenomena work against each other and this ridge represents the point at which the increasing inertias are most dominant. As marked by the vertical blue line, Vertigo's current configuration is at an optimal mass for its pendulum length. It is worth noting that just as the mass variations generate an optimal ridge due to counteracting physical principles, so too does the length variation. However, this ridge corresponds to a length that is within the radius of the sphere and was therefore not a feasible design option and was omitted from these results.

Figure 3.13 shows how the observability matrix's condition is affected by varying Vertigo's design parameters. The range of inverse condition values spans only about .0004, and no maxima or minima are present. Accordingly, the observability is not as critical for design because it is not affected greatly. Just as before, the counteracting principles result in a contoured surface.

The mesh of controllability matrix determinants in Figure 3.14 confirms the results from the inverse condition number. As the length increases, the determinant approaches zero monotonically (for this range). The contour of this surface ensues from the same counteracting phenomena mention earlier.

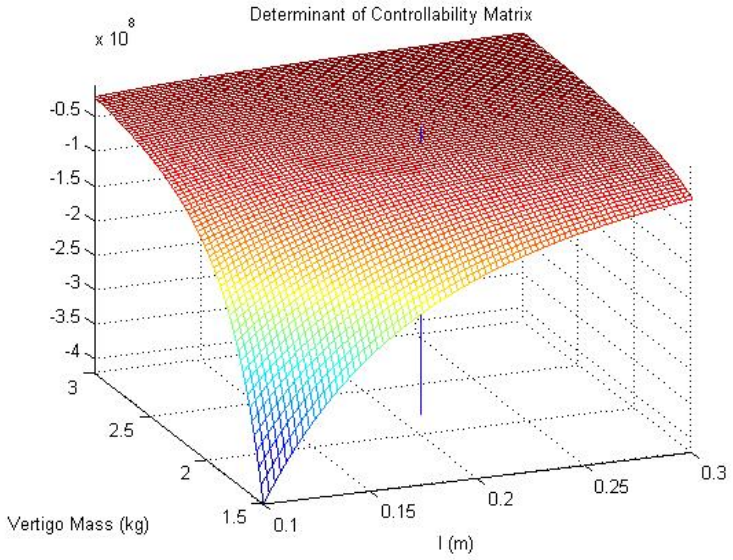


Figure 3.14: Determinant of controllability matrix while varying Vertigo mass and ℓ.

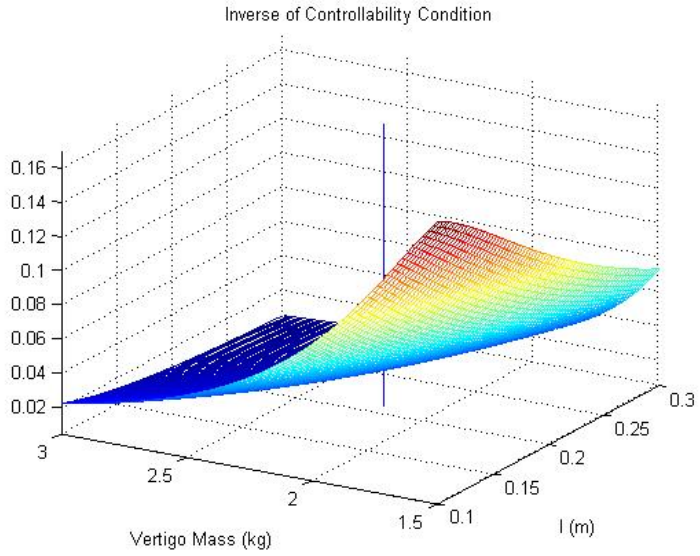


Figure 3.15: Inverse condition number of controllability matrix while varying Vertigo mass and ℓ with fixed rotational inertias.

As proof of concept, this design analysis was also carried out with fixed rotational inertia to isolate the principles at work; Figure 3.15 shows the inverse controllability condition mesh for this. By comparing these results with the original findings, it is clear that these design parameters most significantly impact controllability through rotational inertia when mass is increased. Without the increasing inertia acting to improve controllability, the matrix becomes more ill conditioned with increasing mass. This shows that increasing rotational inertia reduces the required control effort for recovery.

Sphere Design Analysis

In this section, controllability and observability are first checked by plotting the rank of these two matrices while the sphere design parameters, sphere mass and sphere radius, are varied. Consistent with previous plots, the vertical blue line in the following figures represents the current configuration of Vertigo's parameters.

Figures 3.16 and 3.17 show that the matrices have full rank, so it is both controllable and observable for this range of sphere design parameters. Figure 3.18 and 3.19, respectively, look further and show how the condition of the controllability and observability matrices are affected as sphere mass and sphere radius are varied. The controllability matrix is shown to become better conditioned with both increasing mass and radius. This is because as the radius gets larger, the distance from the actuators to the ground increases, and it requires less control efforts to generate the torque needed to move the sphere. This is similar to increasing ℓ. As the mass gets larger, the sphere has more inertia, making it harder to move; however, the control efforts would then have more of a tendency to rotate Vertigo back to the top of the sphere, rather than move the sphere under the body. In practice, there is a balance be-

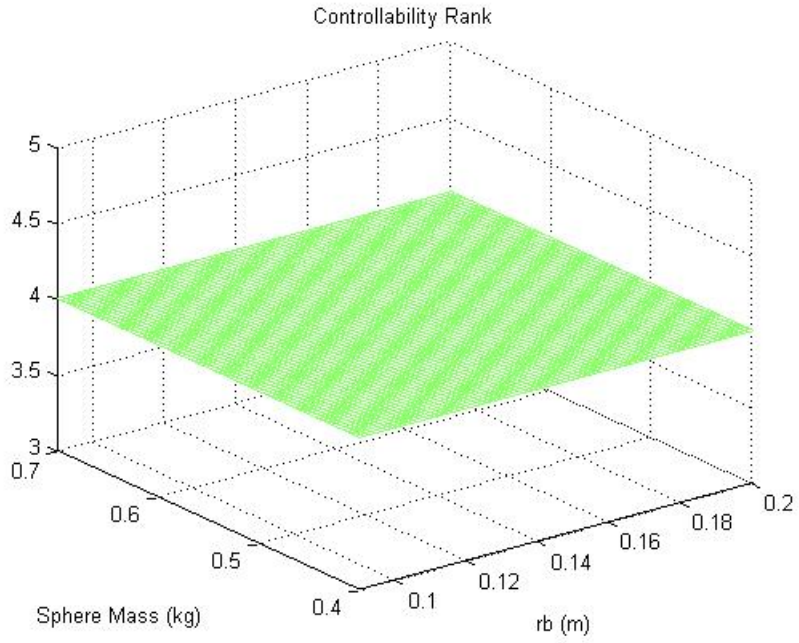


Figure 3.16: Rank of controllability matrix while varying sphere mass and sphere radius.

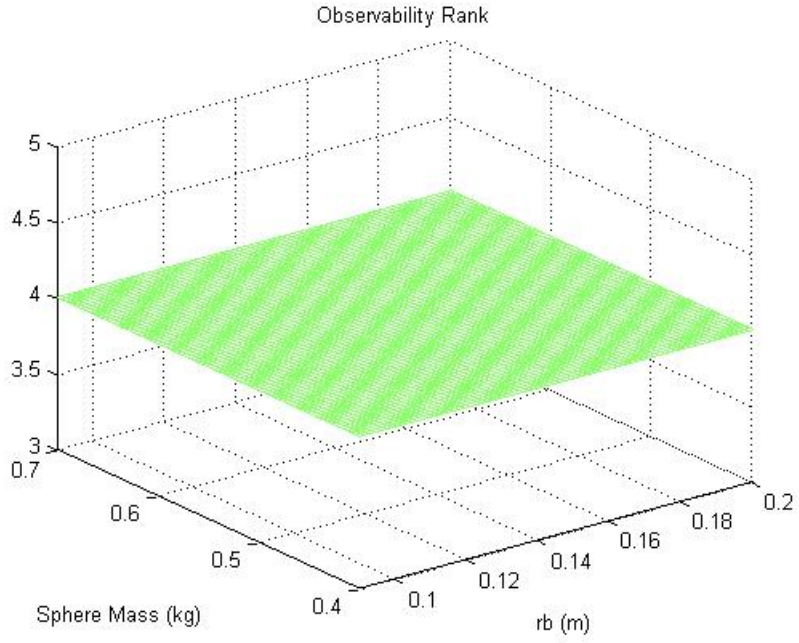


Figure 3.17: Rank of observability matrix while varying sphere mass and sphere radius.

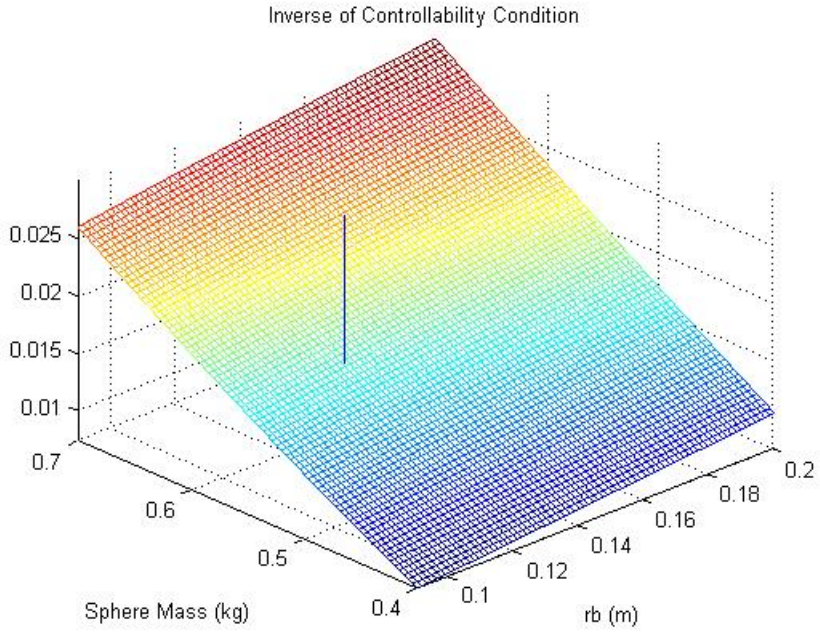


Figure 3.18: Inverse condition number of controllability matrix while varying sphere mass and sphere radius.

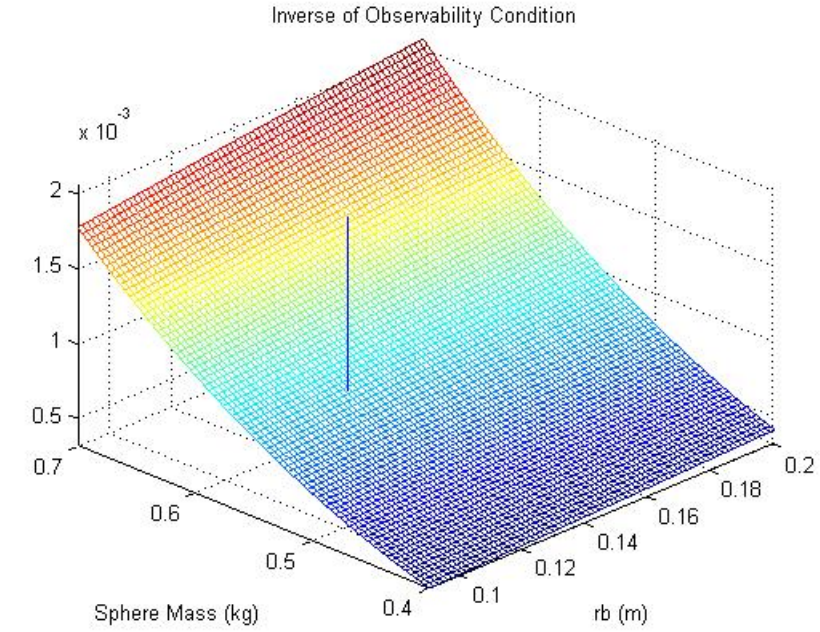


Figure 3.19: Inverse condition number of observability matrix while varying sphere mass and sphere radius.

tween these two, and this is why the plot is contoured, thought it is mild. Figure 3.19 shows that the observability matrix becomes better conditioned with increasing mass and radius. As with the body parameters, the observability condition does not experience any major changes over this considered range; however, the contour with increasing mass is more dramatic then that of controllability.

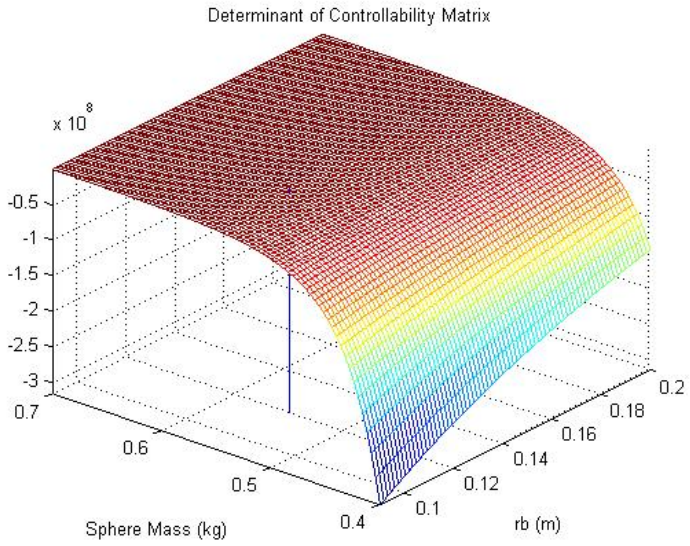


Figure 3.20: Determinant of controllability matrix while varying sphere mass and sphere radius.

The determinant of the controllability matrix mesh is shown in Figure 3.20. It reveals that the determinant is not close to zero, and its highly nonlinear manner of leveling off suggests that it will not approach zero near this range. Therefore, it is safe to assume these results are valid for reasonable sphere properties.

The inertial influences were also isolated for the sphere design parameters; the inverse condition for controllability with fixed inertia is shown in Figure 3.21. The radius of the sphere appears in both body and sphere rotational inertia calculations, and the influence from holding them constant tends to cancel out so no significant

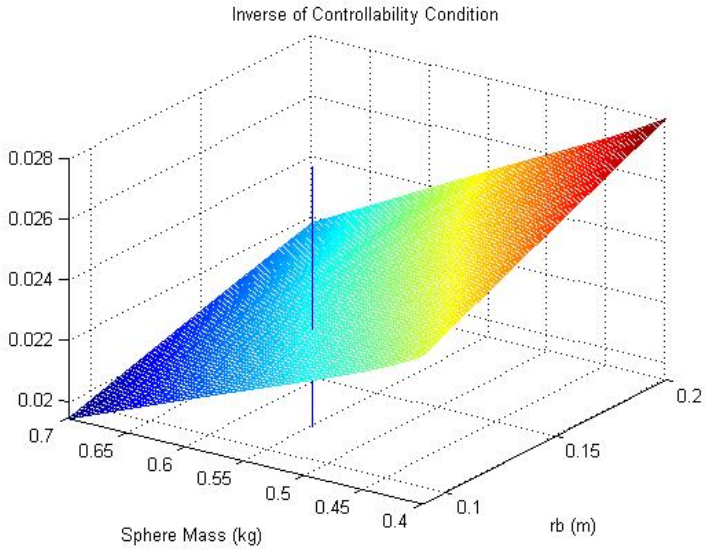


Figure 3.21: Inverse condition number of controllability matrix while varying sphere mass and sphere radius with fixed rotational inertias.

change is observed. However, when the sphere mass is varied, it is clear that the rotational inertia of the sphere was the major contributor to the improved controllability condition because this parameter does not appear in the body inertia calculations.

3.4 Conclusions

This chapter derived the equations of motion for Vertigo, and conducted an extensive analysis to determine their controllability and observability characteristics. It began with the derivation of nonlinear equations of motion for sphere-based Vertigo using the Lagrangian formulation, which followed the derivation of the Ballbot team at Carnegie Mellon University. Their model was chosen based on its accuracy and their success with implementing it. Jacobian linearization was applied to these equations to extract a linear state space model, allowing linear systems analysis techniques to

obtain locally accurate results. The linearized model is also used throughout the control and estimation portions of this thesis. The equations of motion for ground-based Vertigo were then derived. Their simplifying assumptions neglected nonlinear affects, so their linearization was not necessary.

To understand the nature of the sphere-based equations, the controllability and observability of all feasible state conditions was explored and it was found that for every combination of states, the system is both controllable and observable. To get more insightful information about how the controllability and observability are affected by these state combinations, the inverse condition number was analyzed at each point. Smaller inverse condition numbers meant that the matrix is nearing singularity, and computation of its inverse, or the solution to linear systems of equations, is prone to numerical error. The determinant of the controllability matrix was plotted and gave supporting evidence for the inverse condition number analyses. For both controllability and observability, it was found that the most accurate results can be obtained near the vertical, unstable equilibrium. This is ideal because most of Vertigo's motion is confined to this region.

To further explore the characteristic of Vertigo, the controllability and observability analysis that was outlined for the state variations was done while varying physical parameters. These results were used to draw conclusions about the design of Vertigo, and learn what was to be expected for controls and estimation to come. It was found that for the current length between the center of gravity of the body and the center of the sphere, Vertigo is at an optimal mass for controllability condition. The same analysis was conducted for the design parameters while holding the rotational inertias constant and the results compared with the previous analysis. This comparison helped isolate the influence of the many factors that contribute to the controllability of this system. The range of values chosen for the body and sphere parameter

variations were chosen based on physical constraints, yielding an inclusive analysis of feasible options.

Chapter 4

Communication Architecture

4.1 Introduction

In order to implement closed loop control methods on Vertigo, a network of communication had to be established. The main objective of this architecture is to send sensory input to a processor for control law to determine the desired course of action. The control input must then be delivered to actuators to persuade the system accordingly. The response is measured by the sensors and the loop is repeated. This generic schematic can take on many forms, depending on the application, and does so even for the various methods of controlling Vertigo.

This chapter explains the implemented communication methods for onboard sensing and external sensing. Communication for both 1.0 and 2.1 generations of Vertigo was almost identical, so only the 2.1 communication is included here, and acceptations for 1.0 are explained as they arise. Additionally, the sphere and ground-based modes of Vertigo 2.1 had relatively similar communication networks, differing most dramatically in their applied control and estimation; therefore, an instance of ground-based control is used to illustrate the method, and the variations in control and estimation

are covered in their respective chapters.

4.2 Onboard Sensing Architecture

To measure the dynamics of this system, Vertigo was fitted with onboard sensors that were intended to transmit measurements through Qwerk for feedback in the control loop. These devices included the quadrature encoders, Inertial Measurement Unit (IMU), back-EMF and webcam. Despite later discovering that Qwerk did not actually support the majority of these sensors as advertized, they were not omitted from the system because new firmware was being developed by the manufacturer that promised to patch up these connections. Therefore, their physical properties were included in all dynamic modeling to avoid new models having to be derived once Qwerk is updated. This should accelerate the process of incorporating the sensors when they are made functional. Of course, the sensing equipment was also included in the hardware to match the dynamics of the model, this will permit all controllers to be directly applicable after adaptation to fit the new communication methods. Working with Qwerk was a long and finicky process; therefore, none of the control or estimation was hardcoded onboard Vertigo, though it was possible. Instead, the power and familiarity of MATLAB made it the desired program for implementation; therefore, communication had to be established between Qwerk and MATLAB to pass on sensory and control inputs. This in itself was a laborious task and is covered later in this chapter.

For the onboard sensing architecture, Vertigo's brain was rooted in Qwerk, and served as a central hub for receiving and transmitting signals. As seen in Figure 4.1, it received sensory input from the motor EMF feedback, quadrature encoders, camera

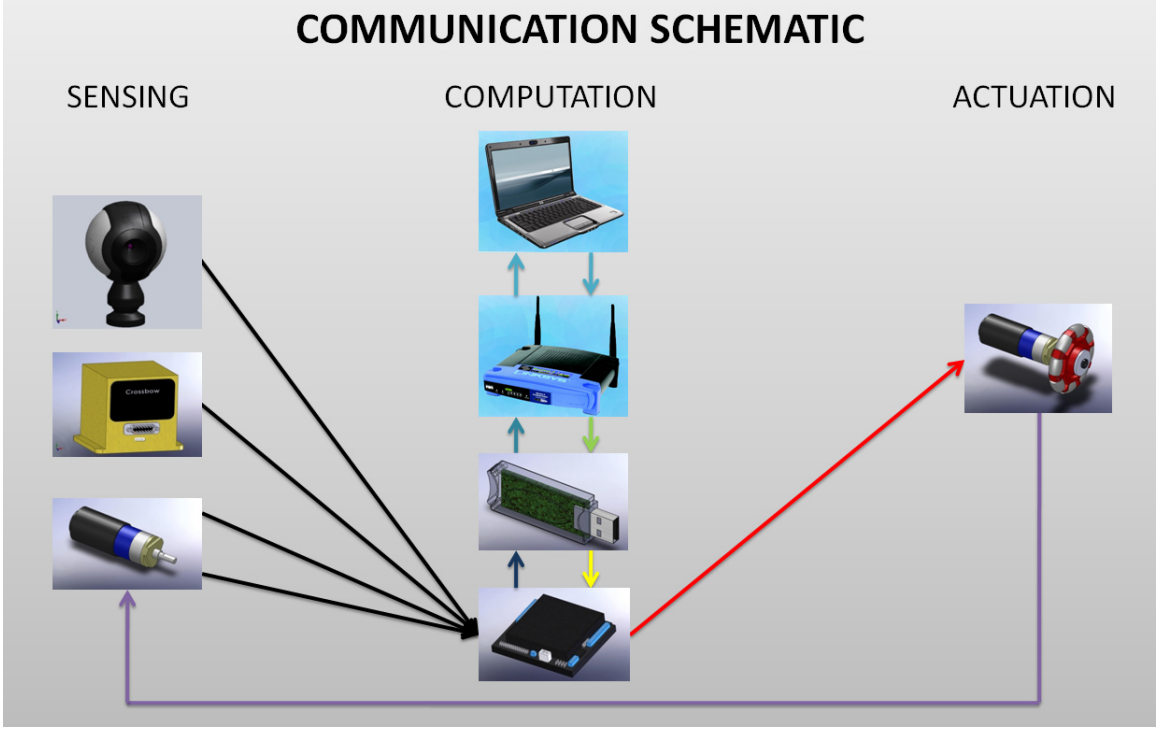


Figure 4.1: Onboard sensing communication architecture.

and IMU, and wirelessly transmitted these signals to a PC where they were taken as inputs in a Simulink or MATLAB control loop. The loop calculated the appropriate control to correct the system's motion and then wirelessly sent it back to Qwerk where voltages were prescribed to the motors. When the system responded, the loop was repeated. The intricacies of these lines of communication were rather complex and will be detailed later; they were spoken of in generalities here because this method had not yet been implemented, leaving no specific details to speak of.

4.3 External Sensing Architecture

All implemented forms of Vertigo's communication architectures enact the same general functions: sense, compute and actuate. They differ mostly in their means of accomplishing these tasks. The schematic in Figure 4.2 depicts a specific instance of ground-based control as an illustrative example of the methodology. To avoid redundancy and excess detail, the major aspects of this description can be applied to sphere-based control as well. Since onboard sensing was not available, Vicon, an external motion capture system, was used for measuring the dynamics. Vicon works by locating retro-reflective markers through triangulation of multiple infrared strobe cameras. The infrared light emitted is a high frequency electromagnetic wave that borders that of visible red light. Working with such short wavelengths allows for highly accurate measurement of the position of each marker. The relative speeds of ground vehicles does not produce a large enough shift for Vicon to register, preventing the Doppler effect from fully being exploited; therefore, all rate states had to be calculated externally with numerical time derivatives.

With Vicon's accompanying software, multiple reflectors can be made into an object, and position and attitude are tracked as such. It is important to recognize that the process by which Vicon defines the object is dependent on the markers; however, the object's height, size, attitude and origin do not necessarily reflect that of the true body. Therefore, the actual center of mass may be slightly misrepresented by the Vicon object's origin. Attitude tracking has a similar concern. When the object is defined, its Euler angles are automatically aligned with the inertial frame of the system. However, the inertial frame is defined somewhat arbitrarily, making orientation of the body for accurate initialization difficult. As a result, the attitude of the body is often misrepresented by the Vicon object as well. These discrepancies

were made small enough to be assumed negligible in most applications of ground-based control by intelligently positioning the markers and with carefully aligned initial conditions. However, the precision exacted by sphere-based control necessitated that estimation techniques be used. Unaddressed, these erroneous measurements plagued the system with implacably whimsical responses that oscillated divergently.

Once an object was created and tracked, its Euler angles and position, with respect to the inertial frame, were transmitted to a manufacturer provided Simulink plant for deciphering. From there, the sim() command was used in MATLAB to call the Simulink plant and fetch the measurements for the feedback controller. Of all processes executed in the control loop, this was the most time consuming, casting it as the leading role in defining the measurement sampling time. In addition, it took an inconsistent amount of time to acquire the data, ensuing a variable sampling time. Furthermore, this communication periodically paused, causing the input to be held constant, and the controller to fail in sphere-based mode. The cause of this lapse was never confidently pinpointed, but its frequency during waking hours and scarcity throughout the night suggested that it was an interference problem.

For ground-based, and the alternate architecture variations, MATLAB can be thought of as the central hub for all communication. It received and sent signals, and was the platform on which controllers and estimators were written and implemented. Figure 4.2 shows how the trajectories were generated in MATLAB, and how errors were calculated with the measurement information from Simulink. For this specific example of control, a proportional controller was applied to the yaw error. The position error was fed into an LQR controller which determined the appropriate control torque to be applied by the motors. Before this control signal could be sent to Vertigo it had to be conditioned. First it was converted into a representative angular velocity for these motors; this was calculated from a calibration chart bundled with the

motor specifications [23]. Next, the input was transformed into the body frame so its orientation matched the Vicon object, which in turn approximated that of the actual body. Finally, the control was converted into representative scalar values recognized by Qwerk. Yaw input did not necessitate transformation into the body frame because its measurements were made in Euler angles. In sphere-based control, the same was true for all associated angle measurements; transformation was not required, only scaling for input to Qwerk. In the above schematic, scaling the yaw input was jointly accomplished by the proportional controller gain. The conditioned translational inputs were then superimposed with the yaw input, a convenience permitted by the unique manner in which Vertigo is actuated. See Appendix A for more on control superposition.

After the control input had been converted into a form that Qwerk was able to implement in the body-fixed frame, it had to be delivered. Forming the link between Qwerk and an external computer turned out to be one of the more challenging aspects of this project, taking the greatest amount of time. Eventually, two variations of communication through Java were successfully established. These were first discovered by others working with the Qwerk unit, and further matured by the TeRK group [24]. The simpler method required some formatting of Qwerk, but essentially launched Java based graphical user interfaces (GUIs) that initiated wireless communication through a router and governed Qwerk's various functions. These programs were primitive in their facilities, but their specialized nature made them excellent for troubleshooting and diagnostic work. The second method expanded on the principles of the first, but rather than operating from a pre-coded GUI, it allowed original Java programs to be written and run directly. This admitted more fundamental access to Qwerk's faculties, but coding was still done in a fairly high level language. JCreator was used as the editing program for developing Vertigo's Java code, and TeRK pro-

vided a small library of commands that controlled the functions of Qwerk. Of these, the most applicable to this project were the audio, battery voltage and DC motor commands.

The audio commands come in three flavors. In the first, a tone was prescribed in Hz and given a duration of time to be played for. This was used extensively to assign audible signals throughout implemented code to help determine its progress. MP3, PCM and WAV audio clips of 15 seconds or less could be sent with the second method. These audio files had to be saved in the associated directory, and were called by name and extension in the function. The third and final audio command sent text and synthesized it into speech, enabling the robot to say any desired phrase. This function was quite entertaining, but also proved valuable when coupled with the battery voltage command. This partnership formed a safety net by sending an audible alert to warn the user to replace the battery when its terminal voltage dropped low enough to impede Vertigo's response. This prevented crashing due to low power. There were initial concerns with using the audio commands that the code would pause whilst the sound was being played; however, there was no trouble in practice, as the code continued to proceed while projecting sound. Additionally, overtaxing of these commands simply built up a queue for the sound bites to be played in order of assignment.

The DC motor commands were the most sought after functions in the library. Their application required some manipulation because they were written specifically to drive a particular two wheeled robot; however, a logic convention was established to independently control all four motors. The functions accepted scalar values between -50,000 and 50,000 to represent the angular velocity of the motors. Since this board was built to work with a myriad of motors, these values had to be defined on a scale and specified as unitless.

To form the link to MATLAB, a third communication method had to be developed. It built on the progress of the first and second methods by dynamically adding a classpath to establish access to the library of Java commands and a generic GUI that provided a means of wirelessly connecting with Qwerk. This GUI also supported the video feed applications that worked with Vertigo's webcam. Making the last connection opened the door for the Java based Qwerk commands to be inserted directly into MATLAB code. To initiate this union, a program was written that dynamically added the Java classpath and opened up the GUI for connection to Qwerk. This saved a variable in MATLAB's workspace that gave access to the Java library. The nature of this variable did not allow m-files that were not in the same directory, or functions to call the Java commands. Additionally, it had "unsavable" material that could not be saved to a MAT-file for reloading in other locations; therefore, all code had to be implemented accordingly. Further explanation and instruction for all three connection methods can be found in Appendix C.

With these connections made, Qwerk was finally able to receive control inputs. Figure 4.2 only illustrated the use of the motor commands for Vertigo 2.1, and differed from Vertigo 1.0's schematic in that all four motors were controlled rather than just two. Qwerk accepted the conditioned control signal as a representative final scalar value for desired motor speed. A trapezoidal angular velocity trajectory was generated and an embedded PID controller applied to track the profile. The output of the PID was converted to voltage and sent to the motors. Back EMF from the motors was measured and converted to a representative velocity to be feed back into the embedded PID controller.

Qwerk used this embedded control method to regulate the inputs to the motors and protect itself from saturation. The downside to this was that the user was unable to directly control the motor inputs. Signals were always feed into the trajectory

generator, and then tracked by the PID controller, this significantly added to the response time. In addition, there was no documentation that addressed, or even mentioned the problem. Therefore, when the motors were found to have an extremely slow response, taking 20 seconds to accelerate between the positive and negative rotational velocity extremes, it took several months to determine that conservative profile trajectories and poorly chosen hardcoded PID gains were the culprits. This egregiously lazy response time was about three orders of magnitude too slow for inclusion in this control loop which cycled at about 25 Hz. This put it in urgent need of mending.

As embedded features, correcting them meant acquiring source code for Qwerk, deciphering the interactions of hundreds of uncommented C files, developing a method for correcting the errors, rewriting the source code, compiling new firmware, deleting Qwerk's existing firmware, uploading the new image, and reconfiguring the connection settings. This was an extremely time consuming and sensitive process that had potential to damage Qwerk and required that a virtual machine be installed to gain root access to Linux. Even after developing a structured routine, this remained a two hour long process when everything worked properly.

The time and high risk associated with this process made it important that the problems be fixed with minimal iteration. Gains that were too high caused extreme actuator saturation and the process became even more risky. The Ziegler–Nichols tuning method was used to safely improve the PID gains, this gave much greater weighting to the P and D gain and almost zero to the I gain [25]. Also, the embedded trajectory generation files were rewritten to include an acceleration parameter that defined the slope of the angular velocity profile. The initial configuration of this code would have had an acceleration parameter of 0.6; this was changed to 200, provoking the original gentle trapezoidal profile into an approximate step function.

For this reason, the PID gains were tuned to best track a step function. The detailed process and instructions for changing the embedded PID gains and velocity profile are included in Appendix D.

Ironing out the kinks of this convoluted communication network meant working with a total of 10 computer programs and five different coding languages. With such a tortuous journey, maintaining dependable communication was problematic throughout implementation. However, the reward of connecting to an external computer was being able to program all controllers and estimators directly from the familiar MATLAB/Simulink environment.

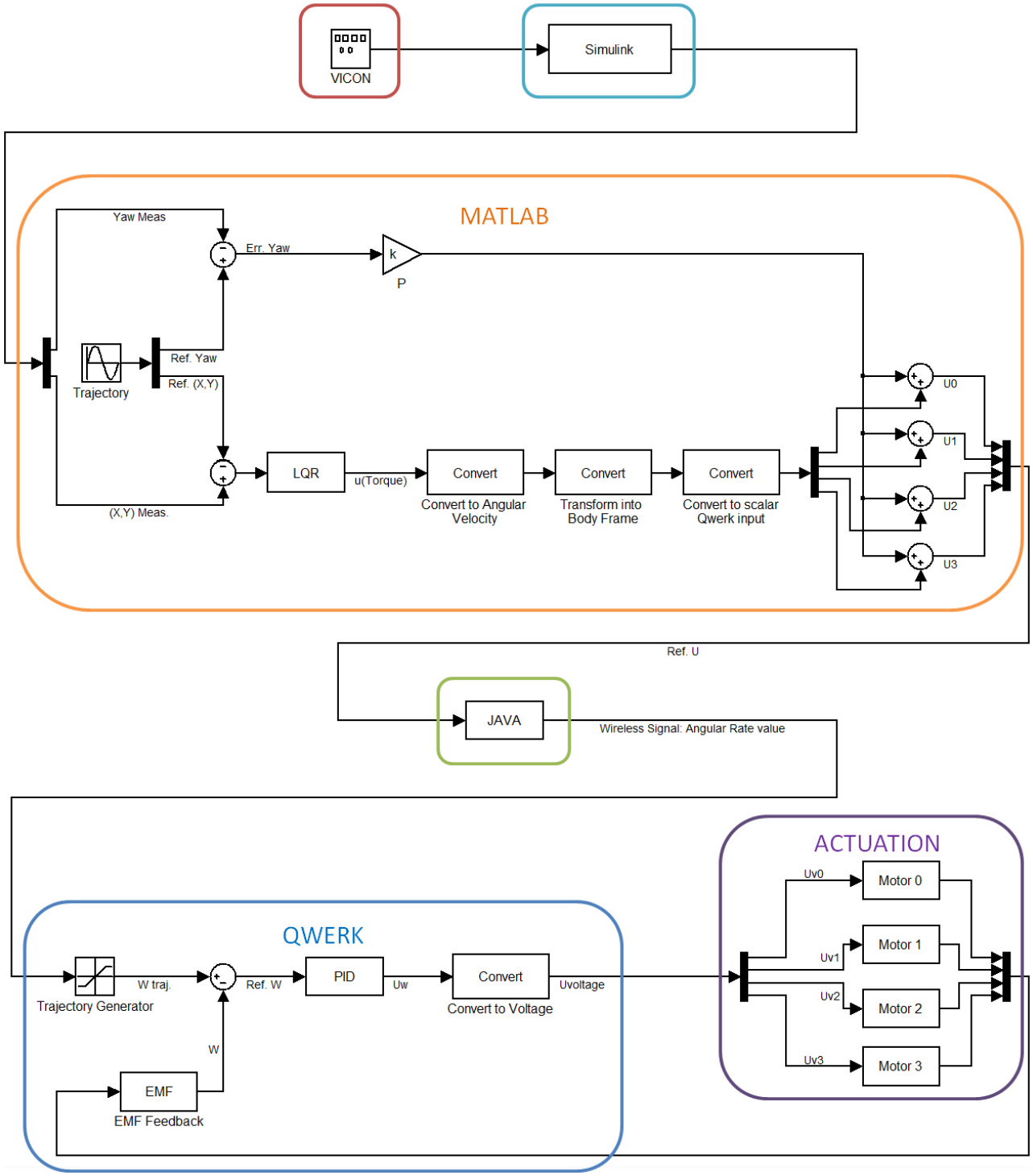


Figure 4.2: Ground-based communication schematic.

Chapter 5

Control

5.1 Introduction

Vertigo is composed of two major constituents: the main body and the sphere. The body includes the processor, power supply, DC motors, sensors and structure. The sphere is the base which Vertigo was primarily designed to balance and navigate on; however, it can also be omitted, and control implemented in a secondary ground-based configuration. Ground-based mode simply means that the body of the robot is placed directly on the ground. In this configuration, the system acts as a more traditional statically stable omni-directional ground vehicle, rather than an inverted pendulum as with sphere-based mode. In both ground and sphere-based modes, the four spherically orthogonal motors drive omni-directional wheels. This actuation method, along with a symmetrical design, provides Vertigo with one of its most convenient features, decoupled control in x, y and yaw. For planar translation, an opposite pair of actuators controls the motion, while the other perpendicular pair slips freely in the direction of motion due to the omni-wheels. Accordingly, the two pairs of actuators are managed independently from one another; however, yaw control

mandates that all four motors collectively work to rotate the robot.

The equations of motion reflect this decoupled behavior, and it is the objective of this chapter to develop controllers that govern the response of these equations. It covers the derivation and simulation of controllers for both ground and sphere-based configurations. These controllers function as verification for the physical concept, and build a foundation for the early stages of implementation by managing the basic aspects of sphere-based motion: balance and translation.

5.2 Ground-Based Control

For the ground-based control of Vertigo, the primary focus was on developing Linear Quadratic Regulator (LQR) control to track a prescribed trajectory. Simulations with this controller were then set up to explore the omni-directionality of the system, and create a standard benchmark trajectory for accuracy assessment on the implemented control. Several other controllers, such as PID, were designed for the ground-based mode, but most of their work was completed as part of their implementation and will therefore be covered in that chapter. For the standard-candle LQR simulation, a consistent measure of accuracy was established as the "norm-error," and a goal for required precision set.

$$\text{norm(error)} \leq 1$$
$$\text{norm(error)} = |\text{norm}([\text{norm(errx) norm(erry)}])\ |$$

This criterion for error calculation accounted for the accumulated error in both x and y position, and in time. It did so by calculating the difference in position from

the trajectory at each time step, normalizing the vectors of x error and y error, then normalizing the vector of these values, and taking the absolute value. Because this value was dependent on both space and time, a standardized trajectory had to be used for evaluating performance to maintain uniform and comparable results. This trajectory was chosen to be a circular path with a radius of 0.5 meters, centered at the origin, that traversed the circumference once at an angular velocity of 0.3 radians/second. A circular trajectory was chosen to study the response of both sets of actuators, and validate the concept of omni-directional motion. In addition, it also gave many opportunities for experimental variation when implemented, such as sinusoidal waves and ellipses.

During the LQR controller development, discrete time equations of motion (with $T_s = 0.04$) were used. These were derived earlier in the Mathematical Modeling and Analysis chapter, and follow the planar model of the ground-based configuration.

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \end{bmatrix} = \begin{bmatrix} 1 & 0.04 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0.0240 \\ 1.1976 \end{bmatrix} T \tag{5.1}$$

$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0 \end{bmatrix} T \tag{5.2}$$

The discrete time LQR tracking problem is designed for full state feedback; however, as seen from the equations, this measurement model does not have such luxuries. In order to obtain full state measurement, numerical derivatives were taken to approximate the rate states for implementation.

By definition, LQR works by forcing the states to zero. In the tracking problem,

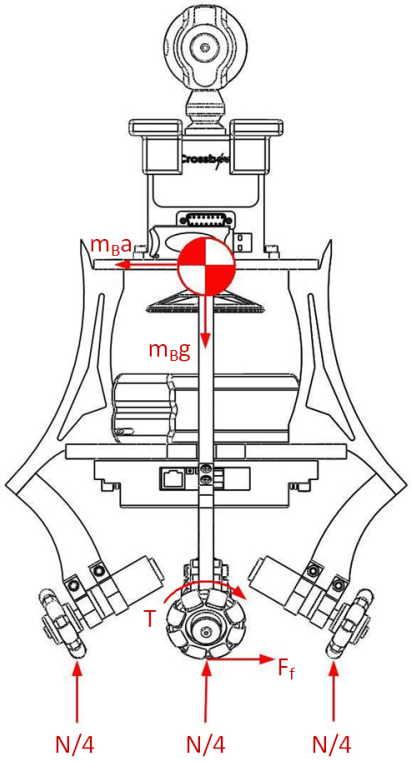


Figure 5.1: Ground-based free body diagram.

it is desired that the states be forced to a reference trajectory, rather than zero. To accomplish this, a new system is defined where the states are the errors between the current and reference positions. Therefore, when the LQR controller acts to minimize the new states, it is minimizing the error, and forcing the system to follow the desired trajectory. The LQR controller was designed by taking advantage of MATLAB's built-in lqrd function. This solves for the optimal controller gains for a prescribed system and cost weighting. The proper mathematical development of LQR tracking is posed as a minimization problem for the performance index J.

$$J = \frac{1}{2}\sum_{k=k_0}^{k_f}\{[C\mathbf{q}(k) - \mathbf{q_r}(k)]^T Q[C\mathbf{q}(k) - \mathbf{q_r}(k)]\} + u^T(k)Ru(k) \tag{5.3}$$

Where, $\mathbf{q}(k)$, $u(k)$ and $y(k)$ are the state, control, and output vectors with lengths of

2, 1, and 2 respectively. Also, Q is a 2x2 dimensional positive semidefinite symmetric matrices, and R is a 1x1 positive definite symmetric matrix. These two matrices are defined to weight the cost function according to the desired performance. It is desired that the error $\mathbf{e}(k) = y(k) - z(k)$ be minimized with minimum control effort, where $z(k)$ is reference trajectory. After solving the Riccati equation, the optimal control input becomes,

$$u^*(k) = -L(k)\mathbf{q}^*(k) + L_g(k)g(k+1) \tag{5.4}$$

where,

$$L_g(k) = [R + B'P(k+1)B]^{-1}B' \tag{5.5}$$

$$L(k) = [R + B'P(k+1)B]^{-1}B'P(k+1)A \tag{5.6}$$

$$\mathbf{q}^*(k+1) = [A - BL(k)]\mathbf{q}^*(k) + BL_g(k)g(k+1) \tag{5.7}$$

$$g(k) = A'\{I - [P^{-1}(k+1) + E]^{-1}E\}g(k+1) + C'Q \tag{5.8}$$

$$P(k) = A'P(k+1)[I + EP(k+1)]^{-1}A + V \tag{5.9}$$

$$V = C'QC \tag{5.10}$$

$$E = BR^{-1}B' \tag{5.11}$$

The above equations show how the matrices Q and R can be viewed as weightings for the state and control minimization. In general, only the ratio of these weightings is of importance, so it is standard procedure to set R as the identity matrix, and only manipulate Q. Once the optimal gains were determined, the command dlsim was used in MATLAB to simulate the response. This simulation was done for both the continuous (using lsim) and discrete equations with similar results; however, one

of the focuses of the ground-based work was to investigate the response when treated as a discrete system, so only those results are included here. Figure 5.2 shows the results of the simulated discrete response for the standardized circular trajectory. This trajectory was specified by assigning $Xref = amp * sin(w * t)$ and $Yref = -amp * cos(w * t)$, where amp is the amplitude and was set to 0.5 m, and w is the angular velocity and was set to 0.3 rad/sec.

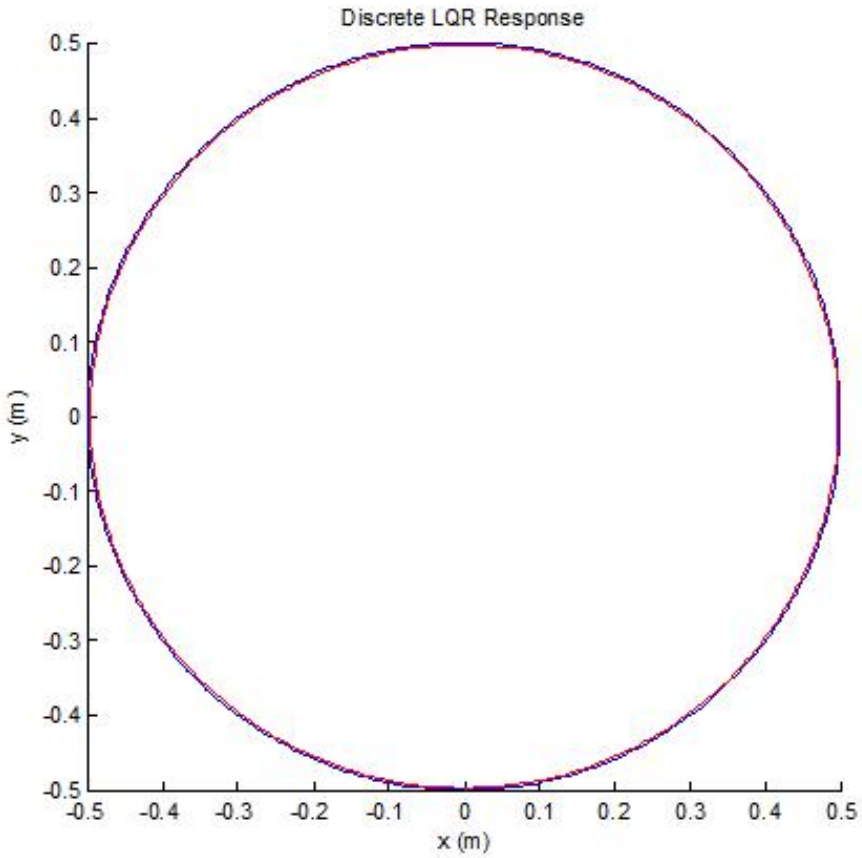


Figure 5.2: Simulation results for a circle reference trajectory: norm-error 0.7821

As seen in Figure 5.2, the simulation results correspond very well to reference trajectory, having a norm-error of only 0.7821. This is within the design criteria of norm-error < 1; however, it was expected that the physical response of the real sys-

tem would have greater error due to the un-modeled, and unpredictable disturbances. This is mostly because the simulation assumes that the yaw angle is perfectly held at zero, without the need for additional control input to keep it there. In reality, this would not be the case because external influences such as slip and uneven ground would accumulate small yaw errors. Also, slip and response lag would further distance the system from the trajectory.

5.3 Sphere-Based Control

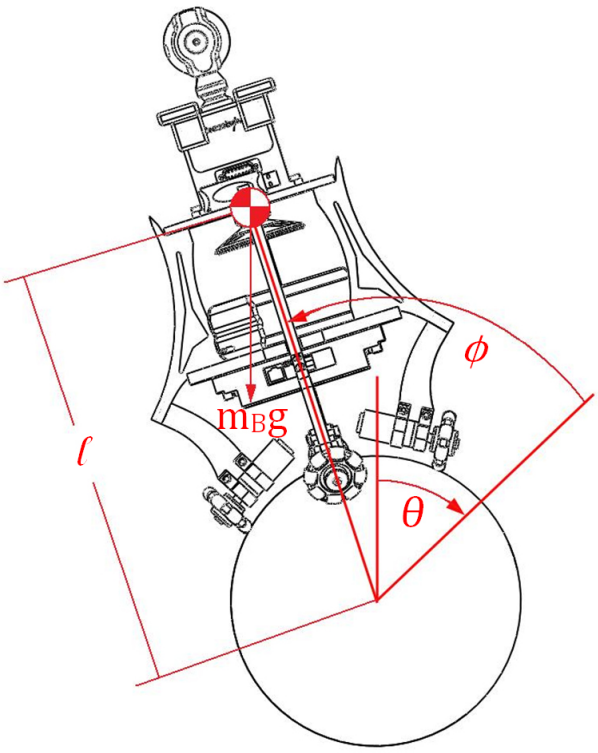


Figure 5.3: Planar sphere-based Vertigo model.

In sphere-based control, the two primary functions are balance and translation. In

this section, several optimal controllers are designed to accomplish these tasks with various criteria for optimization. The techniques that were successfully executed here are for linear systems, so the linearized equations of motion that were derived earlier were used for development. This means that their accuracy is only valid near the unstable equilibrium, which was the point the equations were linearized about. The controllers developed with the linear equations were simulated on the nonlinear equations of motion to ensure their applicability on the real system. The sphere-based planar model, shown in Figure 5.3 is used here, and yaw control is dismissed for the time being. A common condition that many of these optimal control methods require is that the system must be controllable and observable. This was proven earlier in the state controllability and observability analysis for all physically practical state combinations.

5.3.1 Power Optimal Control: *Fixed Final Time, Unconstrained Input*

The first control method considered is Gramian based power optimal control with fixed final time and unconstrained input. It is meant for linear systems and its development begins with the controllability Gramian $\mathbf{W_c}$.

$$\mathbf{W_c}(t) = \int_0^t e^{\mathbf{A}\tau}\mathbf{B}\mathbf{B}'e^{\mathbf{A}'\tau}d\tau \tag{5.12}$$

If $\mathbf{W_c}$ is nonsingular, the pair ($\mathbf{A}$,$\mathbf{B}$) is controllable. It can also be shown that for,

$$\mathbf{x}(t_1) = e^{\mathbf{A}t_1}\mathbf{x}(0) + \int_0^{t_1} e^{\mathbf{A}(t_1-t)}\mathbf{B}\mathbf{u}(\tau)d\tau \tag{5.13}$$

a control defined by,

$$\mathbf{u}(t) = -\mathbf{B}'e^{\mathbf{A}'(t_1-t)}\mathbf{W}_c^{-1}(t_1)\left[e^{\mathbf{A}t_1}\mathbf{x}_0 - \mathbf{x}_1\right] \tag{5.14}$$

will transfer the system from $\mathbf{x}_0$ to $\mathbf{x}_1$ in time t_1. The development and proof of this can be found in [26]. The input $\mathbf{u}(t)$ defines the minimal power control because, for any input $\overline{\mathbf{u}}(t)$, with the same state transfer in the same amount of time, it holds that

$$\int_{t_0}^{t_1} \overline{\mathbf{u}}'(t)\,\overline{\mathbf{u}}(t)dt \geq \int_{t_0}^{t_1} \mathbf{u}'(t)\,\mathbf{u}(t)dt \tag{5.15}$$

The proof of this is well known and can be found in [27].

This method requires *fixed final time* for which the controller is designed, and does not impose any constraints on the trajectory, or input. Figure 5.4 shows the results from a linear simulation for the Gramian based power optimal controller that takes Vertigo from zero initial conditions, to $\mathbf{x}_f = [2\pi \ -2\pi \ 0 \ 0]^T$ in 1.5 seconds. Physically, this means that the controller should move Vertigo from a stationary and upright position, to a distance equal to one sphere circumference away while ending balanced upright with no velocity.

As shown in Figure 5.4, the linear simulation of this implemented controller takes all the states to their desired location in the prescribed time. To do this in only 1.5 seconds is an impressive claim; however, close inspection of the state trajectories and control inputs show that this short time requires very extreme magnitudes and would not be feasible with the implemented system. As stated before, this method

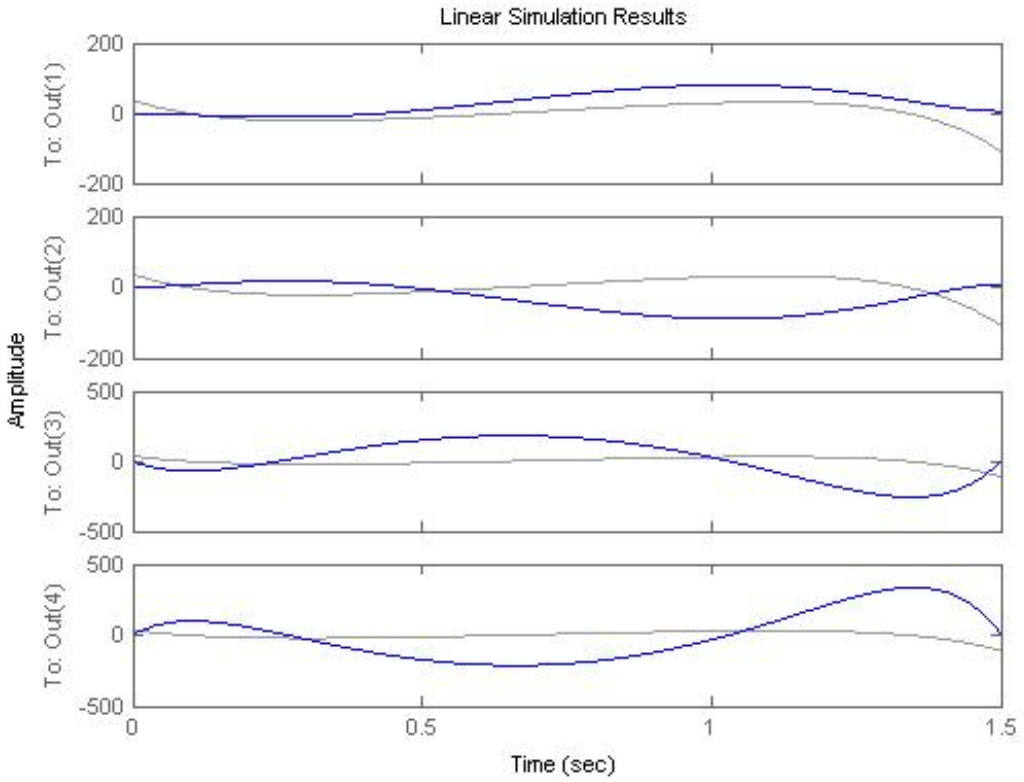


Figure 5.4: Gramian based control for 2π translation in 1.5 sec.

imposes no limitations on input or trajectory, so for situations where final states and time are fixed, and all paths and inputs are available, the power optimal Gramian based controller is a viable option. However, for Vertigo, and in most other real world applications, constraints bind the possibilities for performing a maneuver. To accommodate this, a technique had to be developed to adjust the final time to fit realistic capabilities.

5.3.2 Power Optimal Control: *Free Final Time, Constrained Input*

In most real systems, actuators are limited in their input capabilities; attempting to exceed these limits will lead to saturation and can cause damage to the hardware. For the new motors on Vertigo 2.1, and the capabilities of Qwerk, according to the manufacturer specifications, a realistic limitation on input **u** is about 45 Nm. To account for this, a numerical method was developed that permits imposing limitations on input. It was implemented in the simulation coding to take a user defined maximum control input, and initial guess for final time. It then calculated the associated power optimal Gramian based controller and determined the maximum control required to carry this out. It defined an error between this and the specified allowable value, then updated the guess for final time based on this error and a scaling factors. This is effectively a *free final time* method that finds the minimum power controller to meet the state and input constraints in the minimum time. This was carried out for several specified maximum **u** values, and some interesting results were uncovered.

Figures 5.5 - 5.8 depict linear simulations of the Gramian based control with decreasing allowable u_{max} values. In Figure 5.5, $u_{max} = 55$ and the minimum time that this maneuver was able to be carried out in, while still meeting the u constraint, was tf = 1.2635 seconds, and all of the final states were taken to the appropriate values. Figure 5.6 represents the estimated maximum allowable input for Vertigo with $u_{max} = 45$, and the minimum time that this maneuver was able to be carried out in was tf = 1.3749 seconds, and all of the final states were taken to the appropriate values. In Figure 5.7, $u_{max} = 20$, the minimum time that this maneuver was able to be carried out in was tf = 2.0154 seconds, but the final state values do not meet the specified constraints perfectly.

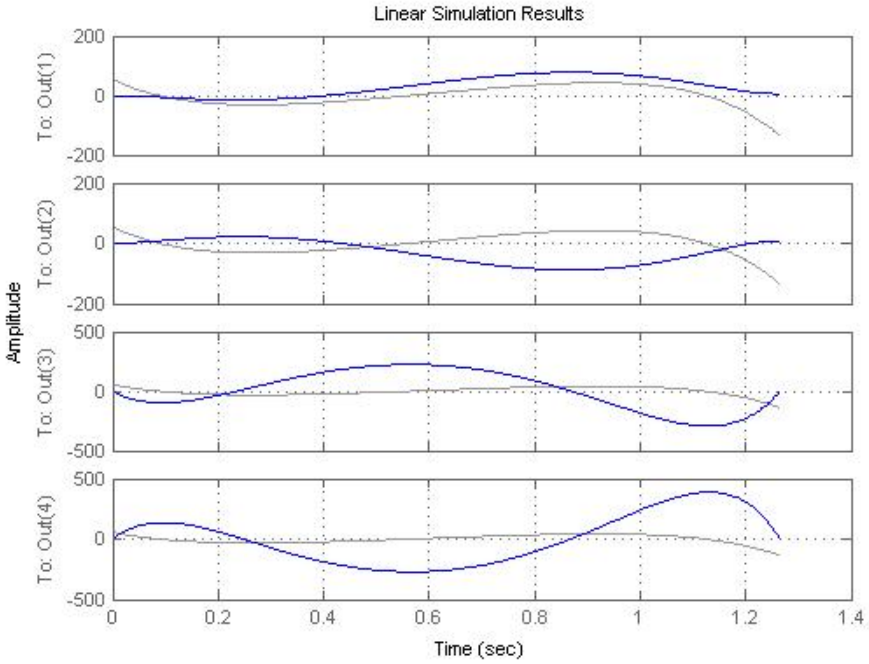


Figure 5.5: Gramian based control, u_max = 55.

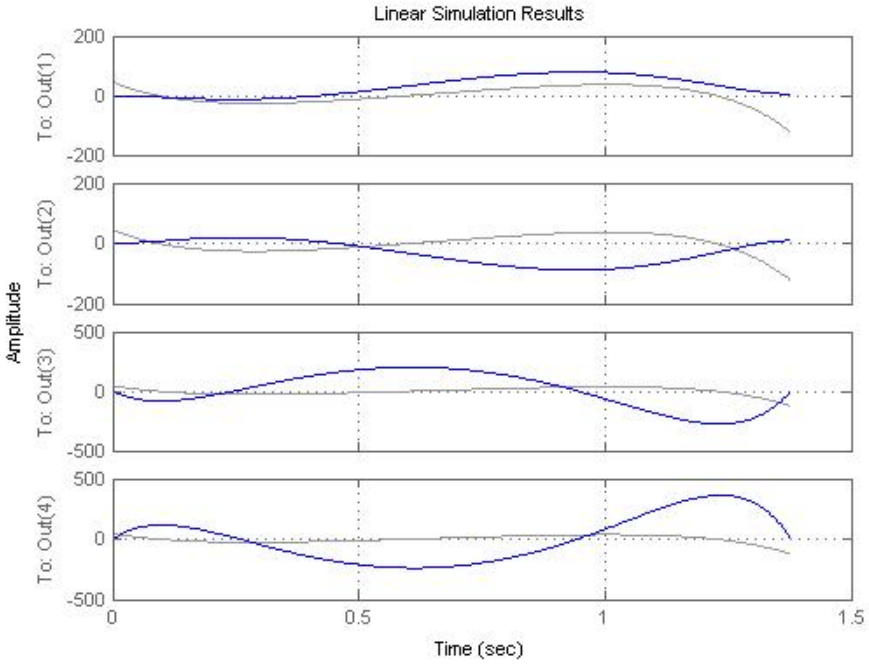


Figure 5.6: Gramian based control, u_max = 45.

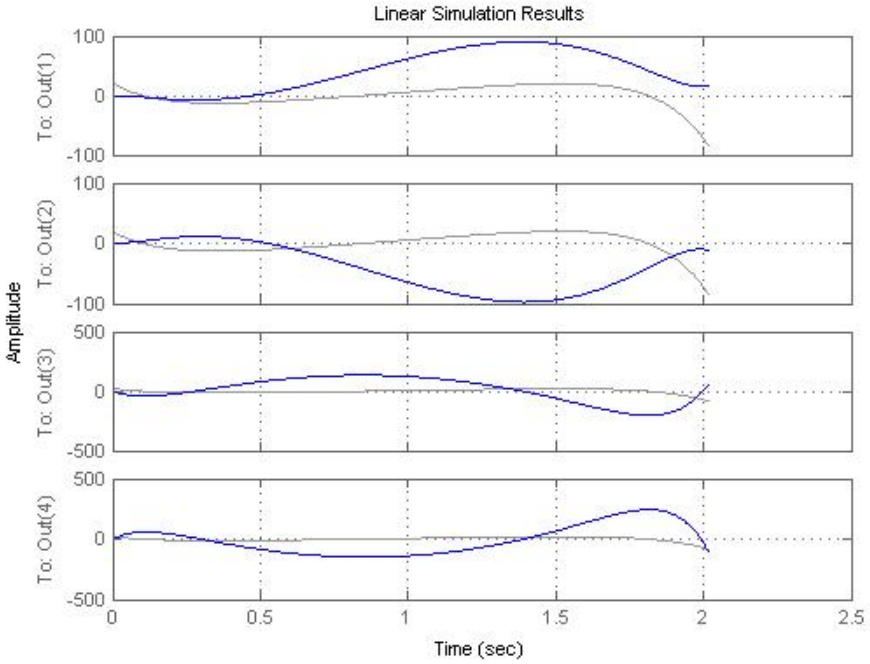


Figure 5.7: Gramian based control, u_max = 20.

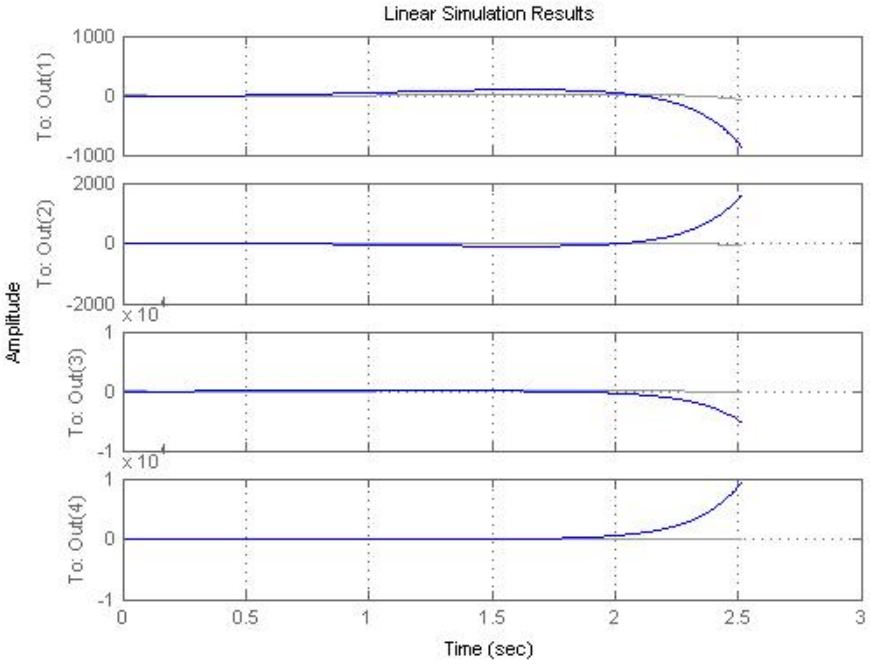


Figure 5.8: Gramian based control, u_max = 15.

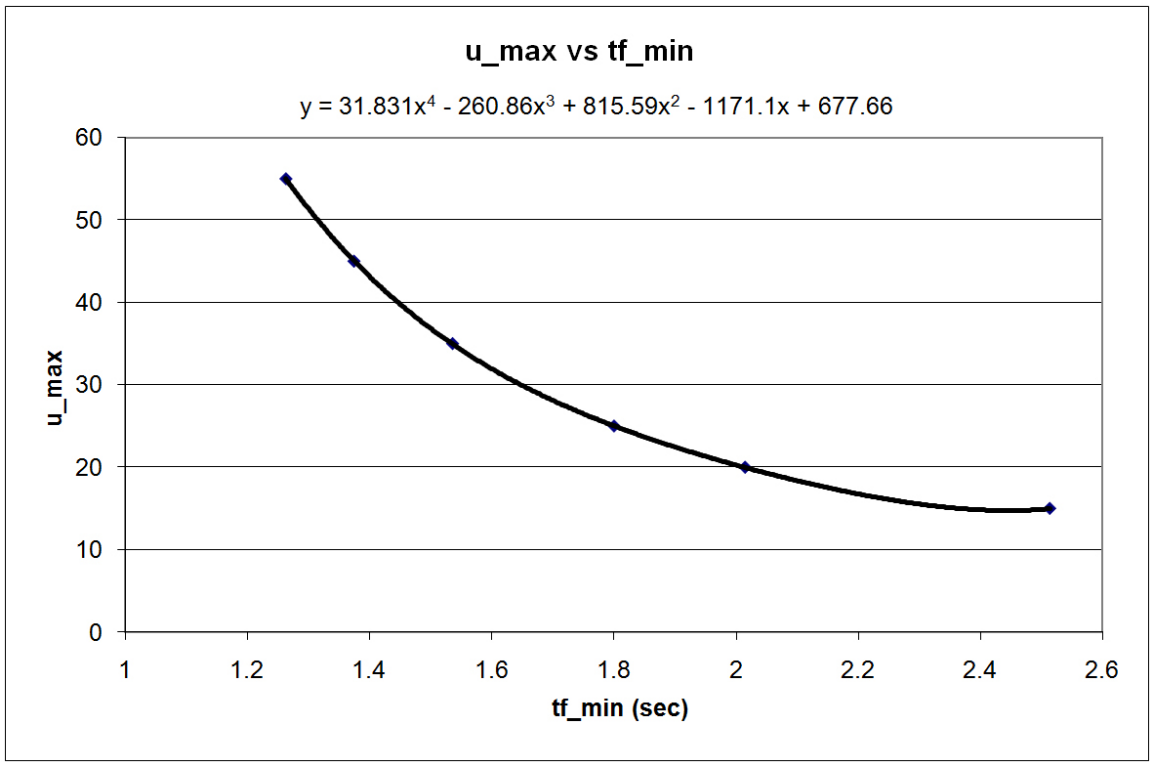


Figure 5.9: Plot of successful u_max convergences and associated final times.

In Figure 5.8, when u_{max} is reduced to 15, the minimum time is tf = 2.5143, but the states do not even approach their specified location. In fact, it was found that for any values below $u_{max} = 15$, the code would not converge, due to singularity errors. Even for $u_{max} = 15$, many attempts and manipulations of the error scaling factor were required before it would converge. The reason is that, for values below 15, the controllability Gramian approaches singularity, and the equation,

$$\mathbf{u}(t) = -\mathbf{B}' e^{\mathbf{A}'(t_1 - t)} \mathbf{W}_c^{-1}(t_1) \left[e^{\mathbf{A}t_1} \mathbf{x}_0 - \mathbf{x}_1 \right] \tag{5.16}$$

is not able to be calculated by MATLAB due to its ill condition. These results support the basis of the inverse condition number presented in the state controllability and observability analysis. Again looking at Figures 5.5 - 5.8, it is clear that, as the

controllability Gramian approaches singularity with smaller allowable u_{max} values, the accuracy of the calculations reduced, and the linear simulation was not able to reach the final states. This error grew as the condition of the controllability Gramian degraded. This also made the code more sensitive to the initial guess for final time, so the plot and included equation in Figure 5.9 were created and used to generate close approximations for the initial guesses based on the empirical results.

5.3.3 LQR Overview

In feedback control, the actuating signal is dependent on both the reference signal, and the output of the system. In state feedback control, the plant feedback output contains at least one of the states, full-state feedback is when all of the states are included as outputs. For state feedback, input $\mathbf{u}$ is defined as follows.

$$\mathbf{u} = \mathbf{r} - k\mathbf{x} \tag{5.17}$$

With this definition, $\mathbf{x}$ represents the output states that are to be fed back into the controller and plant. The state space representation $\dot{\mathbf{x}} = A\mathbf{x} + B\mathbf{u}$ $\mathbf{y} = C\mathbf{x}$ can be rewritten with this input in the following fashion.

$$\dot{\mathbf{x}} = (A - Bk)\,\mathbf{x} + B\mathbf{r} \tag{5.18}$$

$$\mathbf{y} = C\mathbf{x} \tag{5.19}$$

This form reveals that k can be used to adjust the eigenvalues of A. It can also be shown that both the rank and determinant of the controllability matrix do not change with state feedback. This means that controllability is invariant under state feedback, and the previous result showing that Vertigo is controllable in all cases also

holds for feedback control.

Establishing that Vertigo is controllable for feedback control verifies that this method can be used for controller design. To optimally generate the gains, the Linear-Quadratic Regulator (LQR) method was used. In LQR, for a cost function J, where

$$J = \int_0^\infty x^T \mathcal{Q} x + u^T R u \; dt, \tag{5.20}$$

the control $u = -Fx$ minimizes this cost, where $F = R^{-1}B^T P$, and P is found by solving the Riccati equation. From this cost function, it is clear that both the states and input are reaching weighted minimums. These weightings can be altered to best meet the specifications of a given problem, and are often chosen or tuned experimentally when implemented on physical systems. The LQR method poses additional conditions that must be met by the system in order be applied; these are as follows: the linear state space pair (A, B) must be stabilizable, and there must be no unobservable modes. The above state controllability and observability analysis verifies that this system meets these requirements, securing the validity of LQR control for this system.

Two different objectives were considered for LQR controller design to manage the two types of planar motion that Vertigo is capable of. The first was for balance and the second was for point to point translation. The resulting controllers were implemented in a linear simulation to verify their accuracy for theoretically controlling the linear system. When it was established that this method was working properly, simulations were run with the linear control methods on the nonlinear system to mimic the physical system. These nonlinear simulations are the focus of this section.

5.3.4 Stabilizing LQR

For the balance of Vertigo, a linear full-state feedback controller was developed using LQR methods. This process was executed and simulated in MATLAB. Figure 5.10 shows the results of this simulation when applied to the nonlinear system using MATLAB's ode45 integration command. The initial conditions for this, and the remainder of the LQR simulations, are bounded random values to ensure accuracy for all reasonable scenarios.

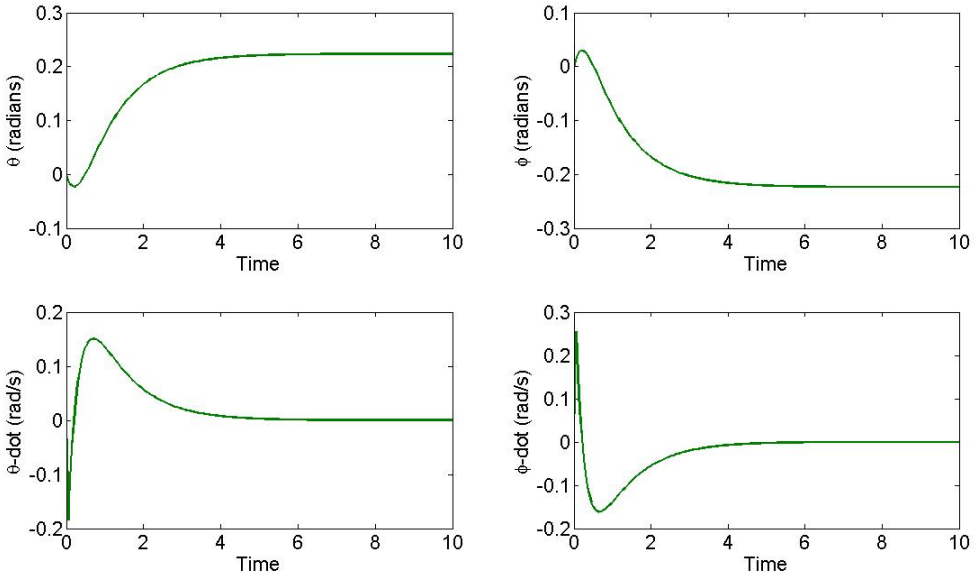


Figure 5.10: State response with stabilizing LQR controller.

It was the primary objective of this controller to balance Vertigo, by bring the states from their initial conditions to ones where the sum of the two angle states is zero. This signifies a vertical and balanced stature. If fully effective, all states should be brought to exactly zero, as this is their minimum possible value. The plots of the rates in Figure 5.10 show that this method was effective at bringing these states back to near zero values; accordingly, the angle states leveled out as the rates were

brought to zero. This controller also attempted to drive the angles back towards zero; however, they over shot and settled about .2 radians away from zero. Offset angle states with zero rate states means that the system settled into a balanced position at a translated location that is away from the origin. Purely in terms of balance, this is not a problem, because the system did not tip over. However, this error may cause difficulty for implementation on the real system because one of the options for the implemented control of Vertigo is to have two separate controllers acting. One of these would be devoted to assuring balance, while the other controls translation. If the balancing controller is doing its job, but causing large translational errors, it will create more work for the second controller and will add to the overall error. Another issue with this controller is that it took this simulation about 4 seconds for the transient response to settle. This is respectable, but reducing this time is desirable if possible because it will further improve overall performance and accuracy.

Though improvement is possible, expecting perfect results is impractical here because this is a linear controller design method being implemented on a nonlinear system. As the states deviate further from the point of linearization, the nonlinear affects of the system have an un-modeled influence on the response. In this simulation, the linearization was taken about a vertical stationary position where all states and control are zero. Therefore, pushing for greater performance increases the magnitudes of the rates and control input which introduces more linearization error. A balance must be reached that provides acceptable accuracy and sufficient performance.

The convention for defining states in this model make it difficult to visualize the motion of the body with respect to an inertial frame. To make this clearer, a plot was generated that sums the two angle and rate state pairs. As seen in Figure 5.11, this definition represents the response of Vertigo in terms of angles with respect to a vertical reference. This clearly shows that the body of the robot returns to

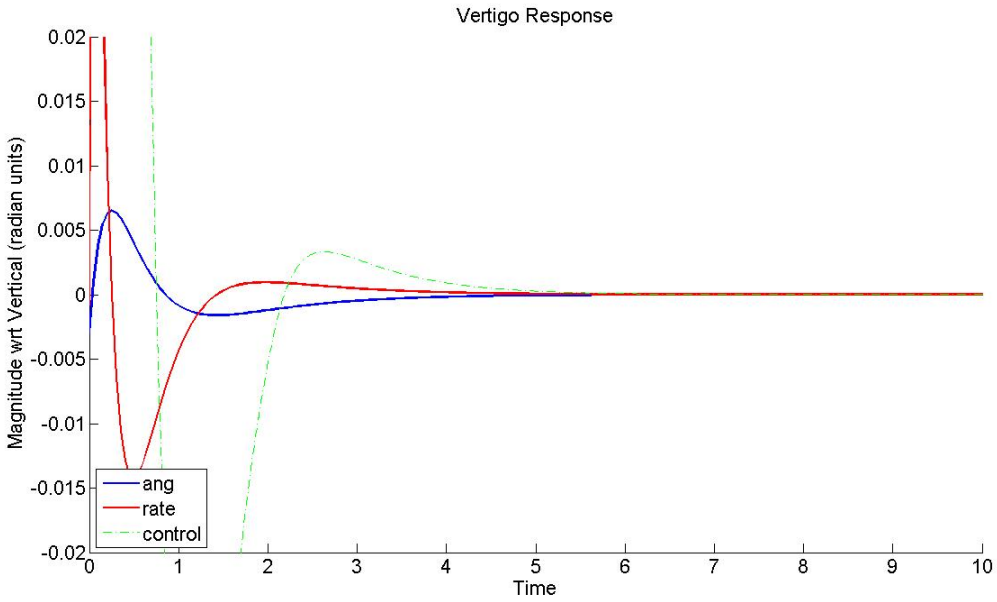


Figure 5.11: Response with stabilizing LQR controller wrt vertical.

equilibrium, but requires about 4.5 seconds to do so.

To mitigate the undesired characteristics, the A matrix was augmented. This effectively includes an additional component to the cost function that continues to penalize the cost with time. Augmentation for this system assumed the following form,

$$A_{aug} = A - \lambda I \tag{5.21}$$

where $\lambda =$ .005 and I is the identity matrix. This augmented A matrix was applied in the LQR procedure to generate new gains. Conservative values were necessary for the augmentation because it is used to manipulate the state space model, which is already a linearized representation of the full equations of motion. Small augmentation prevented extensive departure from the real system and diminished validity of the controller. The Q matrix weightings required retuning for the augmented controller

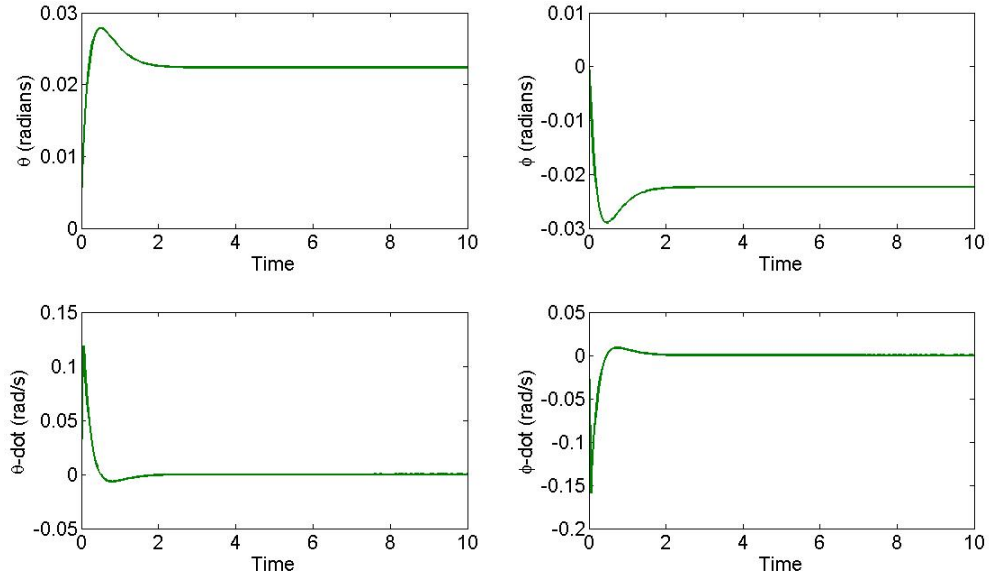


Figure 5.12: State response with augmented stabilizing LQR controller.

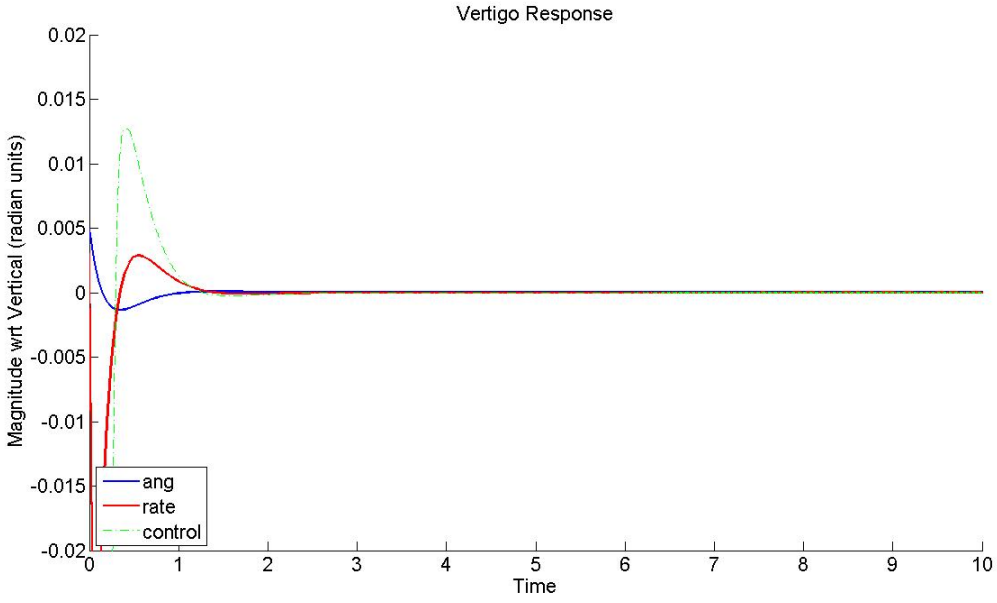


Figure 5.13: Response with augmented stabilizing LQR controller wrt vertical.

design. In the original LQR tuning, equal weighting was assigned to all states; however, the best results for the augmented system were obtained when the angle states had greater weighting to ensure that they were forced to zero.

Figure 5.12 shows that the balancing LQR controller designed with an augmented A matrix was much more effective at managing the states and increasing performance. The steady state bias of the angles was reduced by an order of magnitude, and the rates appear to reach zero much faster than they did with the original controller. Therefore, the augmented controller yields less translational error. Figure 5.13 is the response of the body with respect to the vertical inertial axis. This clearly shows that the body stabilized in under two seconds, which is less than half the original time, and required less control to do so.

5.3.5 Tracking LQR

After balance, the second type of motion that must be controlled for Vertigo is translation. The details of this motion are often conceptually counterintuitive, and are discussed in Appendix A. With some cleaver manipulation, LQR techniques can also be used to develop a tracking controller for translation. By definition, LQR is a full-state feedback method that drives the states to zero; therefore, if the states are redefined as the difference between the current and desired states, the controller will act to minimize the error. In doing so, the current states are driven to the final desired states, affectively tracking the reference states. To implement this, a new control input is defined as,

$$\mathbf{u} = \mathbf{r} - k\left(\mathbf{x} - \mathbf{x_f}\right) \tag{5.22}$$

which makes the state space

$$\dot{\mathbf{x}} = (A - Bk)(\mathbf{x} - \mathbf{x_f}) + B\mathbf{r} \tag{5.23}$$

$$\mathbf{y} = C(\mathbf{x} - \mathbf{x_f}). \tag{5.24}$$

As with the stabilizing LQR, this technique was implemented and simulated in MATLAB on the nonlinear system equations. The simulation takes the system from random initial conditions bounded near zero, to $\mathbf{x}_f = [2\pi - 2\pi\ 0\ 0]$. In other words, it is taking Vertigo from near vertical rest, to vertical rest at a distance one sphere circumference away.

Tuning the Q matrix for translation was more difficult than tuning for balance. This is because the final angle states were far away from the initial conditions, but the rates were not. Therefore, quickly minimizing the angles required large rates, but this fought against the rate minimization. If weighting is placed more heavily on the rate states, the response will be smooth and predictable, but will take longer, as less aggressive rate trajectories will be permitted. Figures 5.14 and 5.15 show how the LQR tracking response with heavily weighted rates followed a very smooth trajectory, but took nearly 20 seconds to reach the final values. The conservative nature of this response also means that less control effort was required; in only two seconds the body was nearly vertical as it slowly crawled to the final states.

By increasing the weighting on the angle states, and reducing it on the rates, the performance can be improved by allowing the angles to change more quickly to their final values. Figures 5.16 and 5.17 show the response with these weightings, and it is clear that the final states were reached more abruptly. However, this improvement came at the expense of a more chattery response and greater demands from the control input.

This is a tradeoff of performance to accuracy and power, and is common in con-

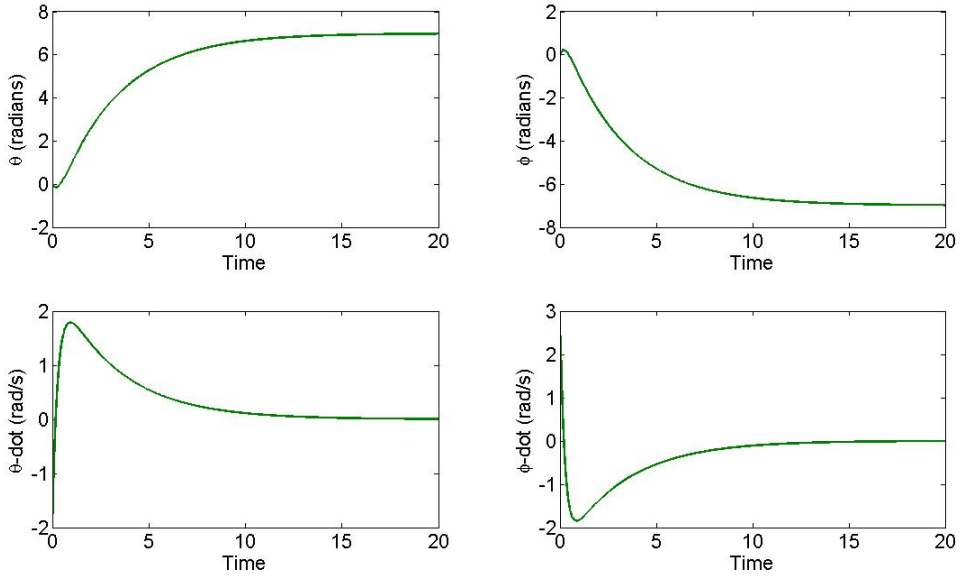


Figure 5.14: State response with tracking LQR controller, heavily weighted rates.

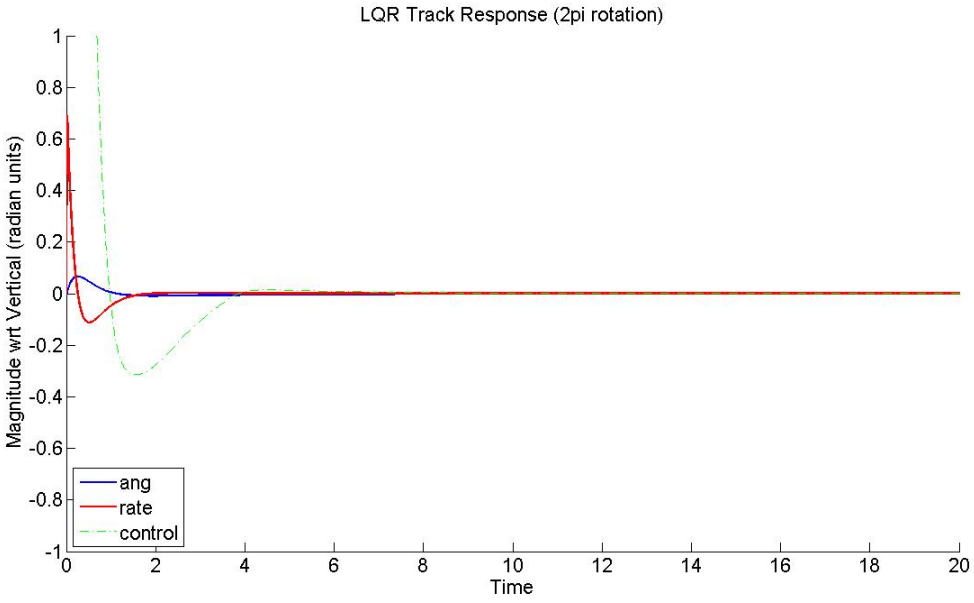


Figure 5.15: Response with tracking LQR controller wrt vertical, heavily weighted rates.

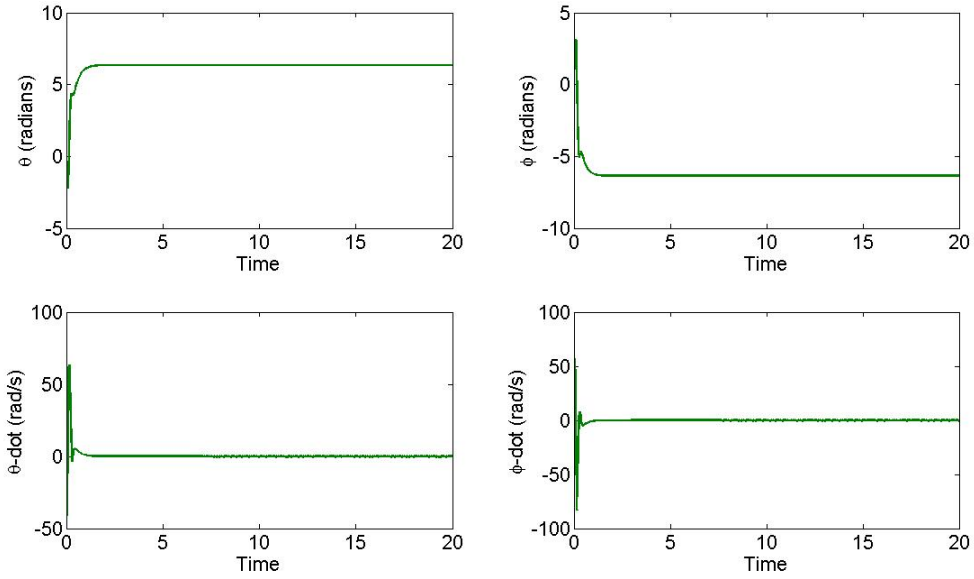


Figure 5.16: State response with tracking LQR controller, heavily weighted angles.

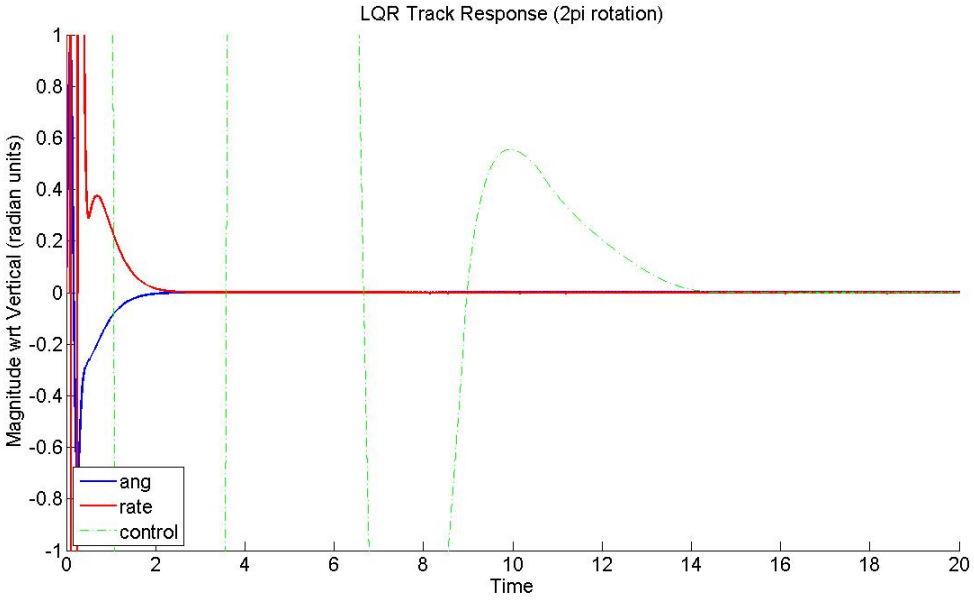


Figure 5.17: Response with tracking LQR controller wrt vertical, heavily weighted angles.

trols. If thought of as a standard transient step response, less weighting on the rates improves the rise time, but increases chatter and control effort. Determining the proper tuning is case specific and finicky. This is where the augmented controller shines, because it is time dependant and helps blend the advantages of both extremes. Low weighting can initially be given to the velocities to improve the rise time, and as time progresses, the augmentation penalizes the rate errors more heavily to smooth out the response. To demonstrate this characteristic, the system was augmented with $\lambda = 1$, and simulated with the same aggressive tuning as the previous simulation. Figures 5.18 and 5.19 show the response of the augmented LQR controller with heavily weighted angle. By comparing this to the two un-augmented responses, it is clear that the augmented response is far superior. It marries the smoothness, accuracy and efficiency of the heavily weighed rate tuning, with the high performance of the heavily weighted angle tuning. In addition, the magnitudes of the control and rates required by the augmented controller are much more realistic than those in the un-augmented response; however, they are still quite demanding.

To represent a more realistic scenario, the same augmented system was more evenly tuned with rate weightings at 75% of the angle weightings. Figures 5.20 and 5.21 show these results. With realistic tuning, this maneuver is executed in about 10 seconds and there are low expectations of the rate and input magnitudes. This simulation is thought to be fairly representative of what should be reasonably expected when implemented on the physical system.

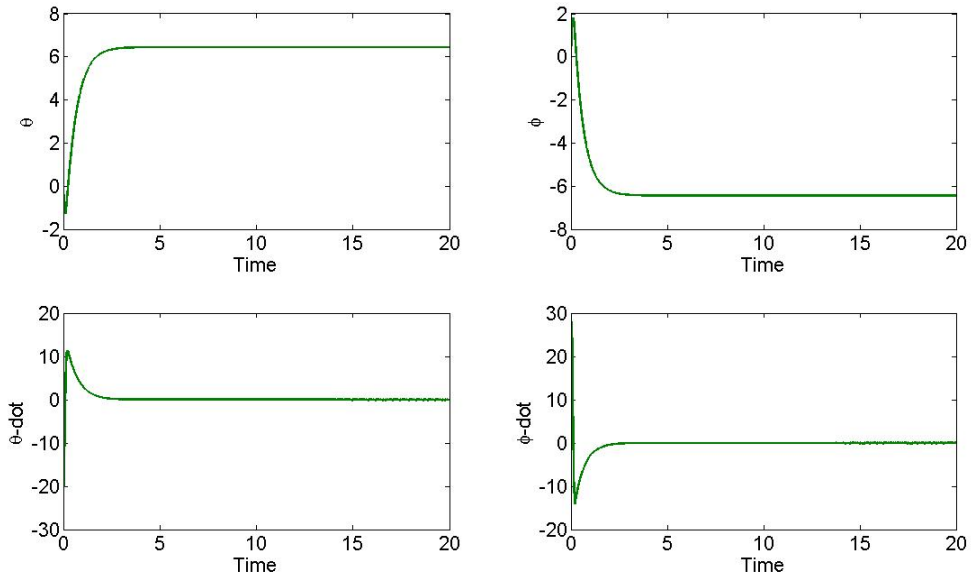


Figure 5.18: State response with augmented tracking LQR controller, heavily weighted angles.

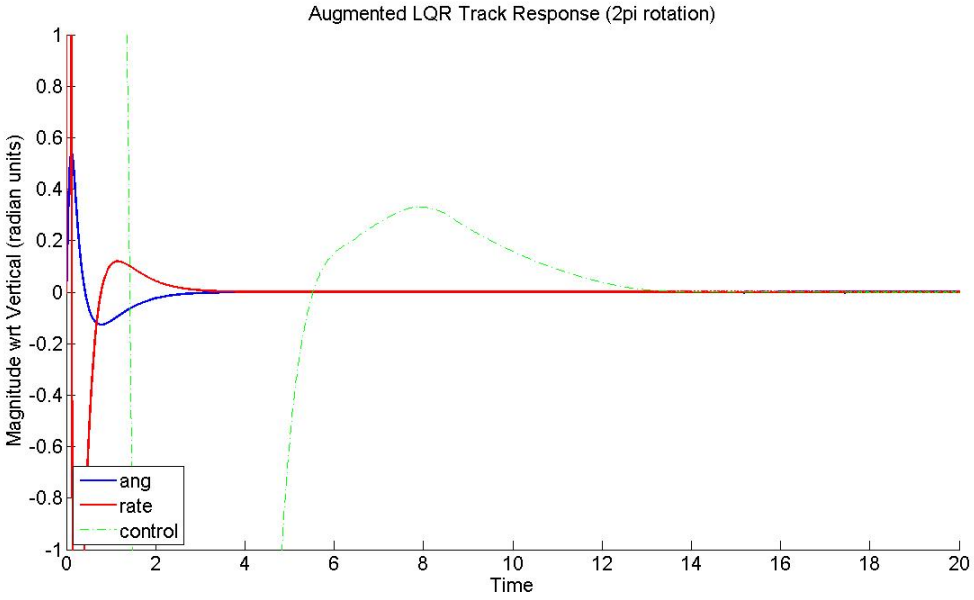


Figure 5.19: Response with augmented tracking LQR controller wrt vertical, heavily weighted angles.

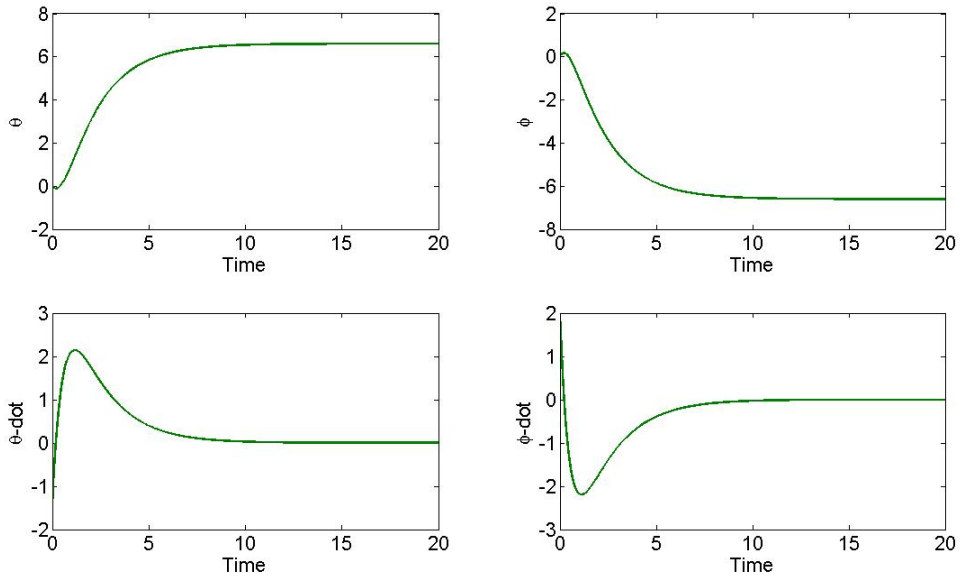


Figure 5.20: State response with augmented tracking LQR controller, realistic weighting.

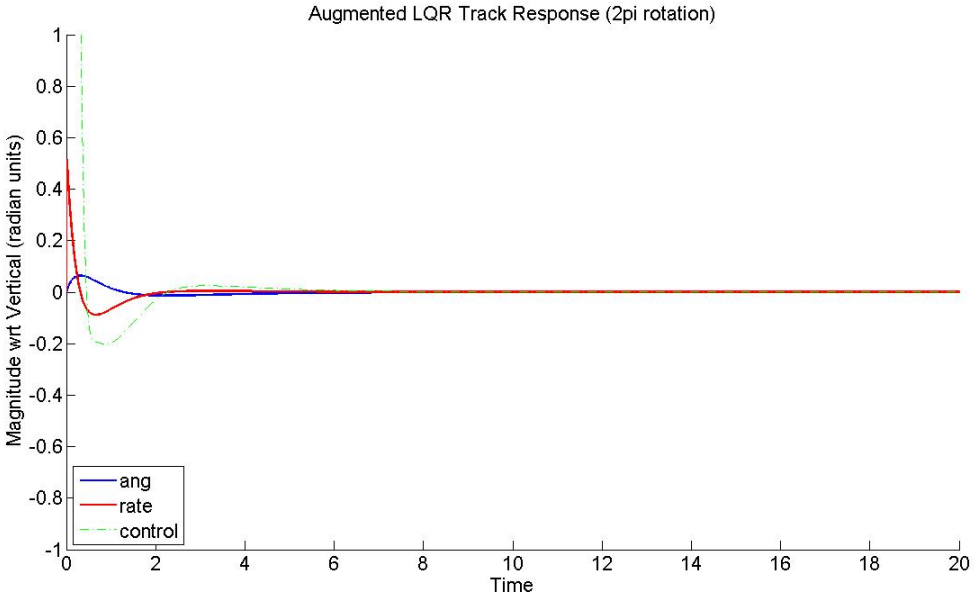


Figure 5.21: Response with augmented tracking LQR controller wrt vertical, realistic weighting.

Chapter 6

Estimation

6.1 Introduction

As an unstable system, Vertigo requires very responsive and accurate control to operate successfully. In developing feedback control techniques, it was assumed that perfectly accurate measurements were available for all states and times; however, this is not realistic when considering implementation on the actual system. Once all functions of Qwerk are made available, information on all states can be measured directly. Until then, the Vicon system had to be used for external measurement, and none of the states are measured directly. Even if full state sensing were available, all instruments are rated with a level of accuracy, which means their measurements are inherently erroneous. The focus of this chapter is to establish a method for extracting reliable state information that can be fed back into the control loop.

Here the continuous-discrete extended Kalman filter was developed and simulated for Vertigo. One of the dominant challenges in controlling this vehicle was the information circuit for closed loop feedback control. The current sensing method utilizes Vicon (an external 8 camera motion capture system) for information on the robot's

position and attitude. Despite the celebrated precision of Vicon, its measurements do not directly correspond to the states of the system. Mathematically transforming the raw measurements to representative state measurements introduces the accumulated uncertainties of the physical system parameters (lengths, mass, etc.), Vicon measurements, and initialization errors. Consequentially, half of the states are not measured at all, and the others only have flawed indirect measurements available. To address this problem, the continuous-discrete extended Kalman filter (EKF) was exploited. This provided a method for full state estimation and noise filtering; however, the EKF requires linearization of the nonlinear measurement model and equations of motion. This added to the uncertainties and had to be included in the development of the filter to mitigate its affects.

6.2 Continuous-Discrete Extended Kalman Filter

The Kalman filter is a recursive process that efficiently estimates the states of a process from noisy measurements. The filter is based on linear dynamical systems that are discretized in the time domain, and is very powerful in that it can estimate past, present and future states while minimizing the mean of the squared error. In this way, it is analogous to applying a standard discrete estimator, but optimally placing the poles with a minimum variance approach.

To be implemented, the filtered process must be modeled in the frame work of the Kalman filter, which assumes a zero-mean Gaussian model for both process and measurement noise. This framework further assumes that the model and measurement noise are uncorrelated from each other and in time. Once modeled, the error covariances and states must be initialized. Based on this knowledge, an optimal expression for computing the estimator gain is applied. State and covariance estimates are then

updated using their current values, the calculated gain, and the measurement model. Lastly, the states and covariances are propagated forward in time to be used as the a priori estimates for the next time interval. The process is repeated for each time step thereafter. This explanation is for the discrete linear system, and follows the original development presented in 1960 when R.E. Kalman first published a paper on the topic [28]. Further development expanded the filters application to other types of systems.

The extended Kalman filter (EKF) is a variation that accommodates nonlinear systems by using their linearized simplifications. When linearizing these equations, the Jacobian is taken, and the point of linearization is reestablished at each time interval with the current conditions. This method is used when applying the filter to Vertigo's nonlinear equations of motion and measurement model to improve and complete data acquisition for its feedback control loop. Vertigo relies on external sensing for dynamic measurements; however, this data contains errors and does not provide direct measurements of the states. There are errors in the model relating these measurements to the states. Furthermore, the mathematical model has errors based on measurement uncertainties and simplifying assumptions; examples of these include the inertias, friction coefficients and the relation between signal input and actual toque applied by the motors. The extended Kalman filter is a well suited tool for estimating Vertigo's states and attenuating the measurement noise. More specifically, this application employs the continuous-discrete EKF because Vertigo is a continuous system, but the measurements are discrete. The full mathematical development of this filter is detailed with its simulated implementation in the next section.

6.3 Application of Continuous-Discrete EKF

This section covers the development of the continuous-discrete extended Kalman filter and its application to the Vertigo system. The general method for this filter is well documented in many sources; for consistency, the convention and notation from [29] will be used here.

To apply the filter, the model must take on the following form.

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}, t) + G(t)\mathbf{w}(t), \qquad \mathbf{w}(t) \sim N(\mathbf{0}, Q(t)) \tag{6.1}$$

$$\tilde{\mathbf{y}}_k = \mathbf{h}(\mathbf{x}_k) + \mathbf{v}_k, \qquad \mathbf{v}_k \sim N(\mathbf{0}, R_k) \tag{6.2}$$

Where $\dot{\mathbf{x}}$ and $\tilde{\mathbf{y}}_k$ are respectively the state and measurement models. f is the system's nonlinear equations of motion, with

$$\mathbf{x} = \begin{bmatrix} \theta \\ \phi \\ \dot{\theta} \\ \dot{\phi} \end{bmatrix} \tag{6.3}$$

as the state vector. $\mathbf{h}(\mathbf{x}_k)$ converts the states into measurement space, and $\mathbf{w}(t)$ and $\mathbf{v}_k$ are zero-mean Gaussian white-noise processes. The G matrix allocates the process noise and is defined as,

$$G = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{6.4}$$

where the first two rows are zeros because they represent kinematical relationships, and it is always true that velocity is the time derivative of position. The equations of motion implemented here are those previously derived for Vertigo. The measurement model will be based on the Vicon data acquisition process until the onboard sensing is supported. To relate the Vicon measurements to the states, geometric and kinematic relations are required.

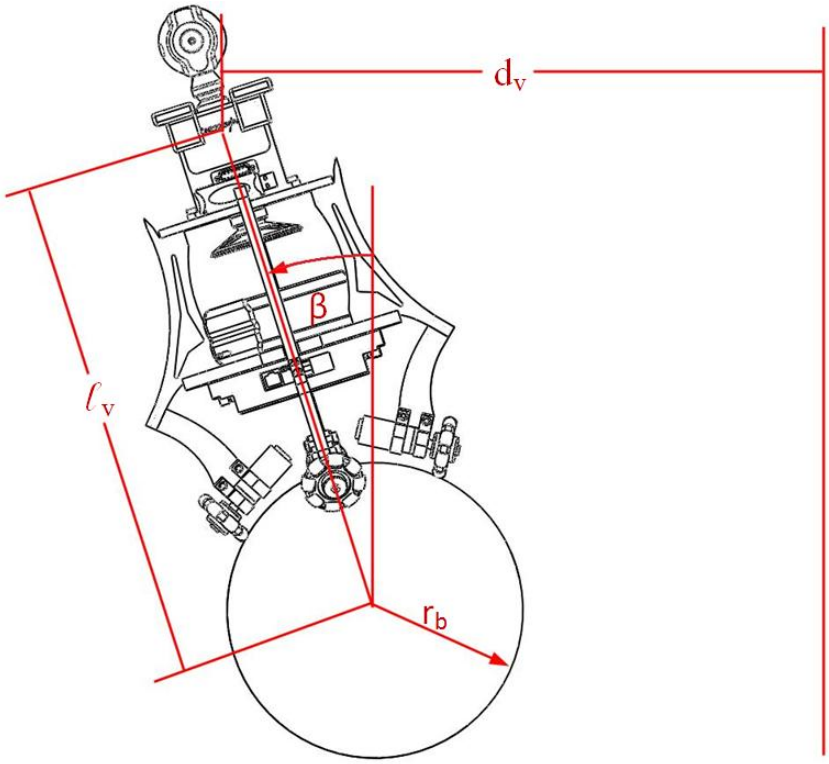


Figure 6.1: Planar measurement model for sphere-based Vertigo.

The Vicon motion capture system works by locating the center of spherical retro-reflective markers through triangulation with multiple infrared strobe cameras. With the accompanying software, clusters of reflectors can be made into an object and tracked as such. The system is then able to determine the position and attitude of this object. All time derivative states must be calculated from these measurements externally. Track the motion of the sphere was not possible with this method because the markers interfered with the actuators and ground. If the entire surface of the sphere were coated in retro-reflective material, Vicon would recognize it as a single marker, and could track its location. However, it would have no inclination about rotation, which is the desired state.

Figure 6.1 illustrates the planar Vicon measurement model. β is the angle of the Vicon object with respect to vertical, and d_v is its distance from the origin. These readings can be related to the states of the system by defining,

$$\mathbf{h} = \begin{bmatrix} \beta \\ d_v \end{bmatrix} = \begin{bmatrix} \theta + \phi \\ 2r_b\theta - l_v sin(\theta + \phi) \end{bmatrix} \tag{6.5}$$

where r_b is the radius of the sphere, and l_v is the distance from the center of the sphere to the center of the defined object.

Vicon is a very accurate system, despite this there is a great deal of uncertainty in these measurements, and it all stems from the way it defines objects. In Figure 6.1, the object is shown to be centered in the upper half of the IMU, this is because the IMU, and its mounting plate, are the most convenient and protected places to attach markers. The difficulty is in knowing exactly how this object is defined. Vicon does their best to simplify the procedure by creating objects with Euler angles that initially align with the inertial reference frame (defined in the initial calibration and setup of Vicon), and measuring distances with respect to the inertial origin. However,

the accuracy of these measurements depends on how well the object is defined with respect to the body it is trying to track, and more specifically, the orientation of its actuators.

Vicon observes all of the markers selected to compose an object and chooses a point to center the object. The exact location of this object with respect to the body is unknown. Similarly, the attitude of the object is defined to correspond with the inertial frame, and is based on the location of the markers at the time it is created. However, if the body is not perfectly aligned with the inertial frame, this object will not accurately represent the body's attitude. Uneven ground and the discontinuous wheels make it extremely difficult to address this misalignment. Together with errors in the radius of the sphere, every term in the measurement model has uncertainty. This means there will be significant errors in the measurement data that will have to be accounted for as best it can by the noise parameter v_k in the Kalman framework.

To commence the cyclic filtering procedure, initial values of the components to be calculated must be defined. For Vertigo, the states and error covariances are initialized as,

$$\hat{\mathbf{x}}(t_0) = \hat{\mathbf{x}}_0 = \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}^T \tag{6.6}$$

$$P_0 = E\left[\tilde{\mathbf{x}}(t_0)\tilde{\mathbf{x}}^T(t_0)\right] \tag{6.7}$$

where the zero initial states represent vertical stature that is stationary at the origin, and $E\,[]$ is the expected value. The Kalman gain is calculated with the formula,

$$K_k = P_k^- H_k^T(\hat{\mathbf{x}}_k^-)\left[H_k(\hat{\mathbf{x}}_k^-)P_k^- H_k^T(\hat{\mathbf{x}}_k^-) + R_k\right]^{-1} \tag{6.8}$$

In this equation, $H_k(\hat{\mathbf{x}}_k^-) = \mathbf{h}_\mathbf{x}(\hat{\mathbf{x}}_k^-)$ is the Jacobian linearized measurement model. Based on the geometry of the model schematic, this works out to be,

$$H = \mathbf{h}_\mathbf{x} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 2r_b - l_v cos(\theta + \phi) & -l_v cos(\theta + \phi) & 0 & 0 \end{bmatrix} \tag{6.9}$$

When a new measurements is available, the Kalman gain can be used to update the state estimate and covariance by

$$\hat{\mathbf{x}}_k^+ = \hat{\mathbf{x}}_k^- + K_k \left[\tilde{\mathbf{y}}_k - \mathbf{h}(\hat{\mathbf{x}}_k^-)\right] \tag{6.10}$$

$$P_k^+ = \left[I - K_k H_k(\hat{\mathbf{x}}_k^-)\right] P_k^- \tag{6.11}$$

If a measurement is not available, the estimate remains, and is used as the initial condition for propagation forward in time. The propagated values are taken as the a priory estimate for the next step, just as the initialized values were for the first step. The equations to be propagated are,

$$\dot{\hat{\mathbf{x}}}(t) = \mathbf{f}(\hat{\mathbf{x}}(t), \mathbf{u}(t), t) \tag{6.12}$$

$$\dot{P}(t) = F(\hat{\mathbf{x}}(t), t)P(t) + P(t)F^T(\hat{\mathbf{x}}(t), t) + G(t)Q(t)G^T(t) \tag{6.13}$$

Where $F(\hat{\mathbf{x}}(t), t) = \mathbf{f}_\mathbf{x}(\hat{\mathbf{x}}(t))$ is essentially the A matrix from the linearized equations of motion. These covariance and state equations are nonlinear, so they must be integrated forward in time. Since the covariance and state propagations are coupled, the two equations must be integrated simultaneously. This was done using the ode45 function in MATLAB by combining both systems. Once propagation is carried out,

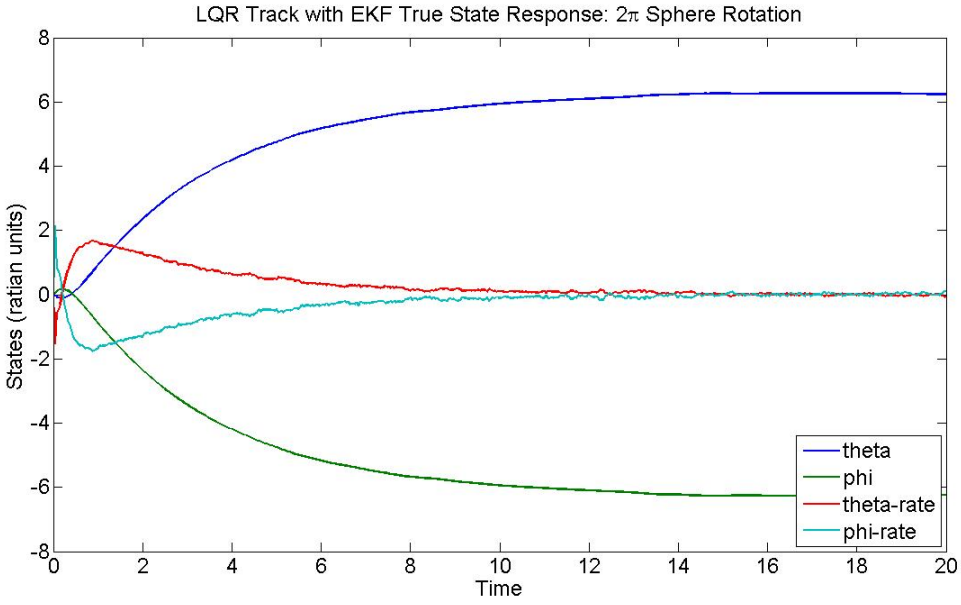


Figure 6.2: C-D EKF, true state response.

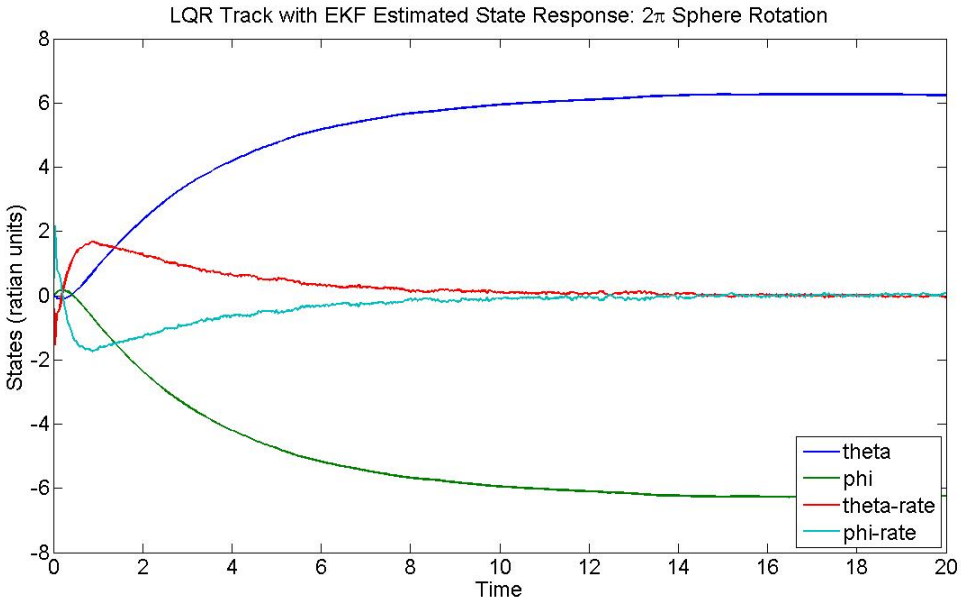


Figure 6.3: C-D EKF, estimated state response.

the cycle is repeated in the next time step.

This filter was constructed and applied in simulation to ensure it was properly incorporated. Not all uncertainties are readily determined in the continuous-discrete nonlinear problem, so tuning of the Q and R matrices was required to improve the estimator results after it was applied. This was a sensitive procedure for the extended Kalman filter because factors such as linearization errors are not easily quantified, so discerning their influence took time. A baseline for this tuning was formed with knowledge of the noise in the process and measurement; from there, refinement was done by trial and error.

In the implemented simulation, a linear-quadratic regulator (LQR) was used for control. Together, the LQR and Kalman filter solve the famous linear-quadratic-Gaussian control problem. The simulation generated the true data for the response of a translation that equates to a 2π rotation of the sphere, the true data is calculated in the filter loop and only computes to the next time step. This is so it can use the estimated states to calculate the LQR control, which mimics how the loop will be implemented on the real system. Noise was added to the true data to generate measurements, and the filter was applied to them. Each time through, the filter calculates its own control using the same LQR method that the true data was generated with. For those simulations that run the filter between measurement updates, the most frequent measurement was stored for plotting at that step.

In the first simulation, the sampling time was based on the actual average sampling time of the Vicon MATLAB implemented communication loop, 0.04 seconds (25 Hz). The states and covariance were directly propagated forward to the next measurement, so every time step updated the estimate with measurement data.

Figure 6.2 and 6.3 respectively show the true and measured response to the simulated continuous-discrete extended Kalman filter. From these results, it is clear

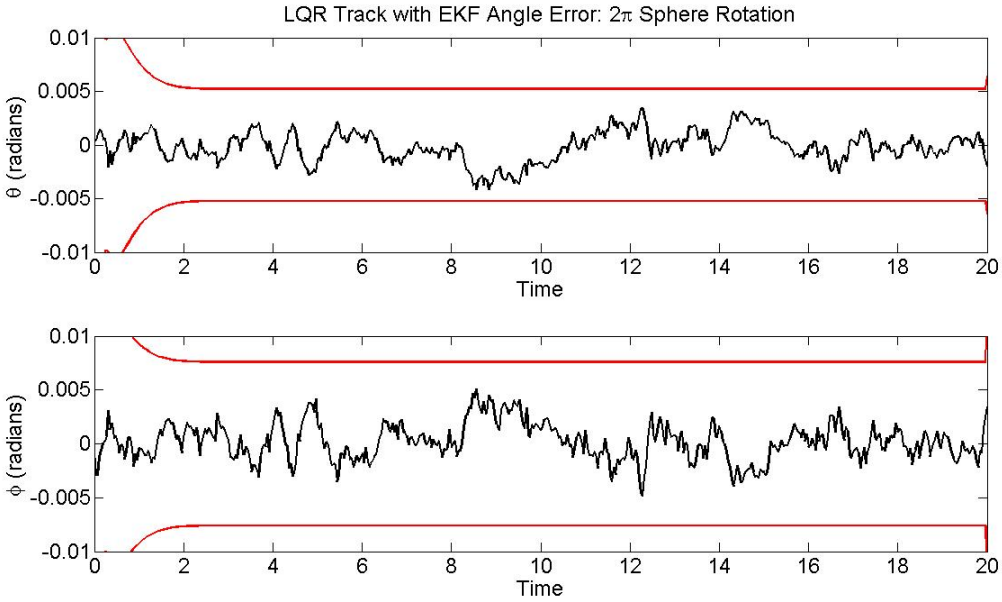


Figure 6.4: C-D EKF, angle state errors.

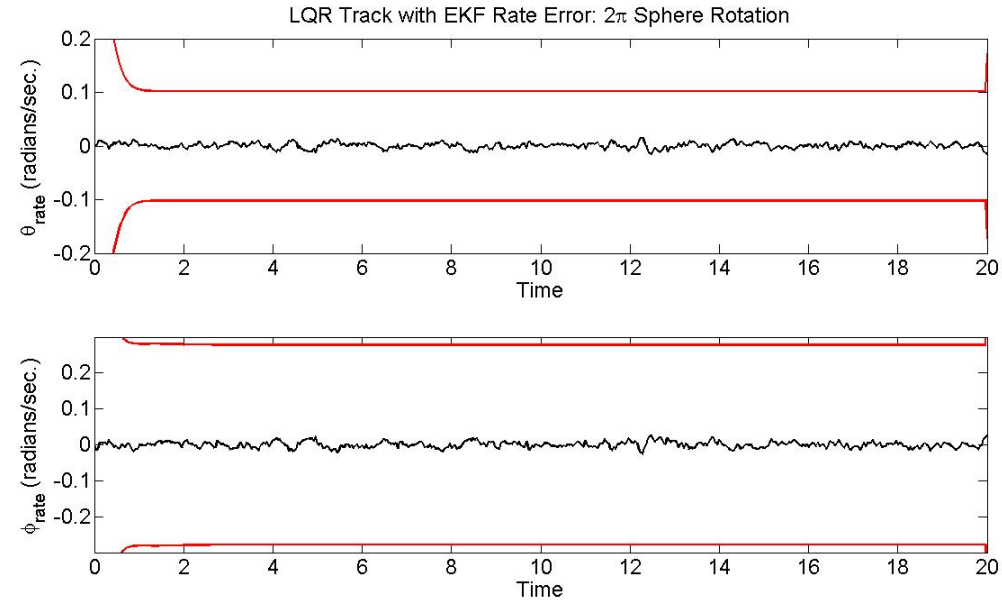


Figure 6.5: C-D EKF, rate state errors.

that the estimated response has effectively represented the true response and filtered the noise. For further comparison, the errors between the true and estimated states were plotted with their respective 3σ bounds. Figures 6.4 and 6.5 show that the errors were small and well contained by the bounds; therefore, the filter was well tuned and succeeded in improving upon the measurement data.

6.4 Sampling Time Analysis for C-D EKF

In some applications, if the measurement sampling time is well known, the filter can be run multiple times between measurements. The objective of this is to try and improve the accuracy of the estimate without requiring more measurements. When executed, the update portion of the filter is not used unless a measurement is available at that time step. In the absence of a measurement, the propagation from the previous step is used as the current estimate, and the filter proceeds as normal. The applied code was set up to separately define the measurement and output frequency. To investigate how this may affect the estimate, a simulation was run using the same realistic measurement frequency of $f_m = 25Hz$, but with an increased output frequency of $f_s = 100Hz$, making the filter run four times per update.

Figures 6.6 through 6.9 show that propagating four times per measurement update, versus only one, in the continuous-discrete extended Kalman filter also provides improved estimates over the measurement data. In fact, close comparison between the two simulations shows that propagating multiple times not only improved the estimate, but slightly improved the transient time of the true response as well. Improved performance is the goal of incorporating estimation techniques. It is important to note that this simulation mandated retuning to converge with the increased output sampling time. Modified tuning also affects the performance of the filter, so this

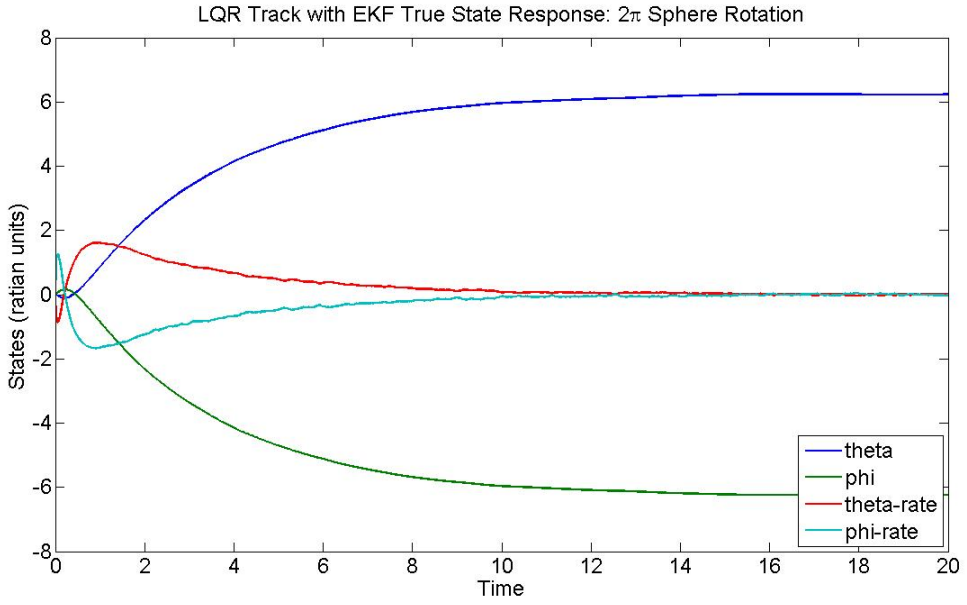


Figure 6.6: C-D EKF, true state response, $f_m = 25Hz$, $f_s = 100Hz$.

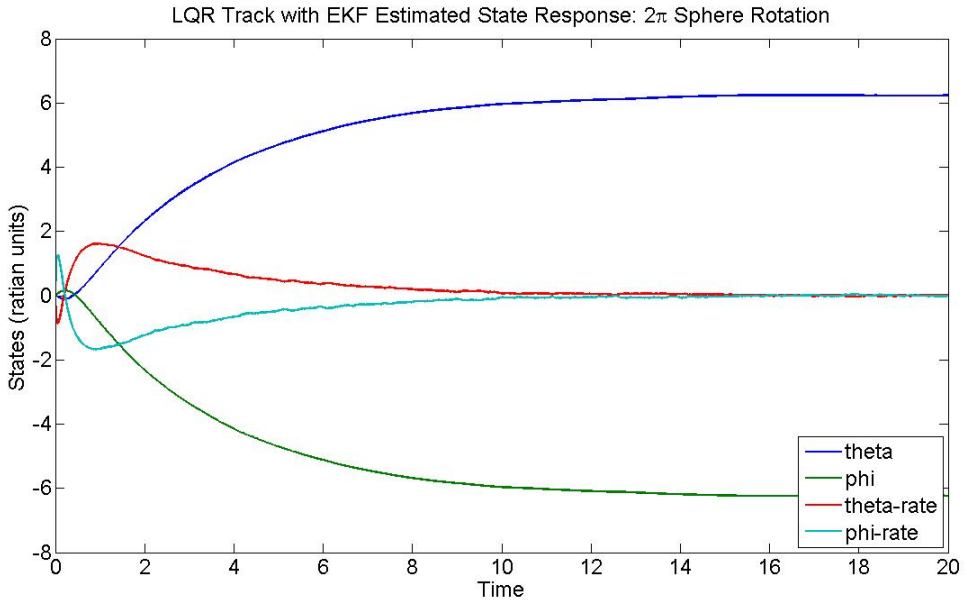


Figure 6.7: C-D EKF, estimated state response, $f_m = 25Hz$, $f_s = 100Hz$.

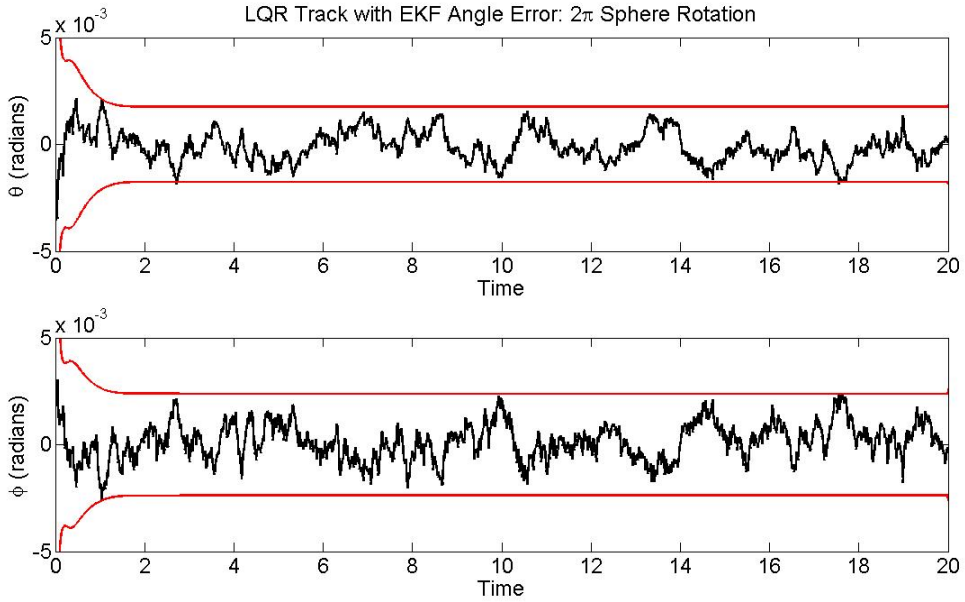


Figure 6.8: C-D EKF, angle state errors, $f_m = 25Hz$, $f_s = 100Hz$.

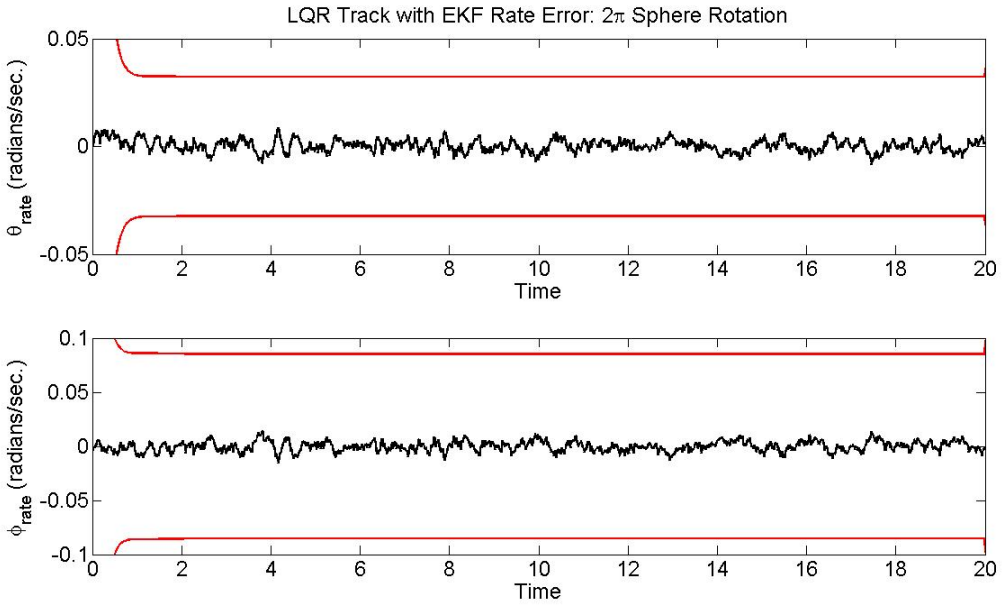


Figure 6.9: C-D EKF, rate state errors, $f_m = 25Hz$, $f_s = 100Hz$.

comparison shows that, when both filters are properly tuned, it is advantageous to have $f_s > f_m$.

It is clear that the number of propagations influences the filters accuracy, but the measurement sampling time also has a significant impact. Understanding the nature of this affect is important for maximizing the chances of success with the fully implemented system. Therefore, a simulation was created that varied both the measurement and output sampling time. As a measure of accuracy, the 3σ values for the states were stored once it had settled for each combination of sampling times. The values were taken 5 time steps from the end, rather than exactly at the end, because this is a cyclic process, and breaking that to exit the simulation causes the last 3σ values to be invalid.

Figure 6.10 is the mesh of the 3σ values for θ as the sampling times were varied. Analysis of all four states yielded the same trends; therefore, only the results for θ were included here as a representative set. The previous analysis showed that increased output frequency improved the estimator; this is confirmed here, though it is not obvious. There are several reasons for this, the most pronounced being that its impact is outshined by that of the measurement frequency variation. Also, the maximum and minimum values for both sampling times were chosen to encompass practical values, and ones that could all be simulated with the same tuning. This fixed tuning was acceptable for all points; however, if it were possible to specify ideal tunings for all simulations, the influence of varied output frequency would have been more apparent. Lastly is the affect of diminishing return. Running the filter with $f_s = 2f_m$, will improve the results, increasing that to $f_s = 4f_m$ will show further improvement; however, a limit exists to how accurate a filter can be without receiving more frequent measurements. Because the range of variation was restricted by tuning, it was not possible to start this analysis from the baseline point of $f_m = 25Hz$,

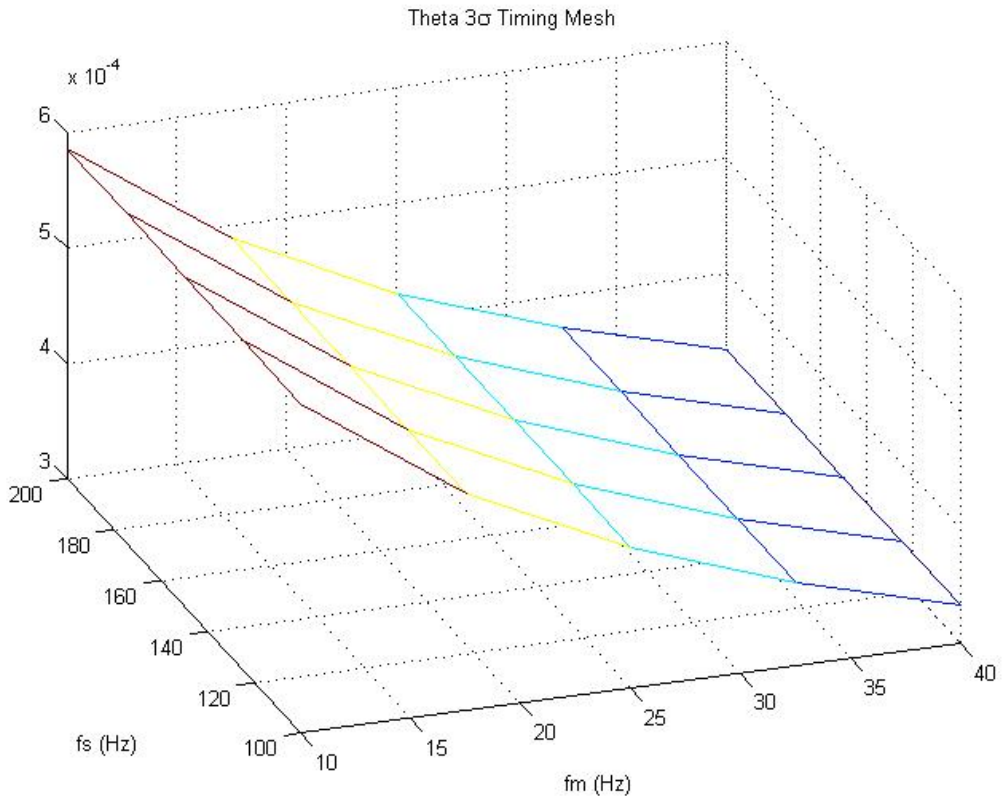


Figure 6.10: Sampling time mesh.

$f_s = 25Hz$, and expand from there. It was also not realistic for the two ranges to overlap, because impractical points would exist where $f_s < f_m$. Therefore, it was necessary that $f_s > f_m$ for all points, so the more significant initial improvement of running the filter more than once per update could not be captured.

It is clear from Figure 6.10 that there is a much more influential dependence on the measurement sampling time. Lower 3σ values tend to signify a more accurate estimate. Therefore, this mesh verifies the expected results that the estimate will improve as the number of measurements is increased. Again, the same general results were found with the other three states, so they were not included here to avoid redundancy.

Chapter 7

Implementation

7.1 Introduction

Vertigo is an original platform that was built from scratch; as such, it was necessary to increment its development by introducing controllers from the ground up. Taking small steps built a familiarity with the system that made troubleshooting easier as the project progressed to more sophisticated coding. The goal here was to develop methods for implementation, and apply fundamental programs for testing functionality and theoretical principles. These would serve as a foundation for future work in balancing and navigating on the sphere. In many ways, early implementation mirrored the efforts of establishing the communication architecture. Accordingly, the first controllers were implemented through the Java based Terk programs, and interfaced through the provided GUIs. Complexity grew as feed-forward programs were coded in JCreator. These programs performed simple tasks, and were written to help troubleshoot hardware. This practice developed an understanding of the library of commands that accessed Qwerk's functions, and would later be used to send input signals to Vertigo. This Java interface was also used to calibrate measurements and

develop conversion factors for signals between components.

When it was discovered that the current version of firmware on Qwerk did not actually support the onboard sensors, it was necessary to establish the external sensing communication loop that tied in Vicon, Simulink, MATLAB, Java and Qwerk. The variable stored for connecting to Qwerk did not allow controllers to be implemented in Simulink; therefore MATLAB had to serve as the central hub for coding. With most of the kinks ironed out from this network, basic control implementation began. Many challenges threatened to jeopardize implementation; several were anticipated, but the majority of them were unforeseen. These will be discussed in the remainder of this chapter along with the progression of ground-based control, and balancing control. Finally, attempts at sphere-based balancing control will be detailed with a discussion of what prevented their success.

7.2 Implementation Challenges

Uneven Terrain

Vertigo has four support contacts in ground-based mode, so the slightest curvature in terrain, or inconsistency in actuator dimensions, will lead to uneven weight distribution amongst the wheels. This has several negative effects on performance, all of which pertain to friction. Friction is needed between the wheels and ground to move the body. If a leg is not supporting any weight, it has no normal force to generate its maximum frictional force. This is the primary cause of slip. When this affect is large or prolonged, only one pair of motors will be properly applying control, inducing undesired yaw. If both pairs of motors are slipping, time response will suffer from the wasted control. When tracking a step, slip near the final destination may prevent the robot from achieving the final destination. This is because error is small,

so the applied control was insufficient to overcome the potential causing slip.

Inconsistent Loop Sampling Time

Vertigo's loop of communication calls upon five different programs, two of which require network connections, one being wireless. A delay in any of these systems will cause inconsistent loop sampling time. Many implemented control methods make the assumption of constant sampling time; this introduces error. The effects of erratic sampling times were most detrimental to implementation of the Kalman filter, and will be discussed later in this chapter. To best mitigate this problem, an average sampling time was used and assumed to be constant. There are some cases where the Vicon measurement system momentarily paused, and control was held constant. This caused tracking controllers to deviate from their desired trajectories, and loss of balance in sphere-based mode.

Dynamic Instability

Vertigo was intentionally designed with a high center of gravity to improve its controllability and dynamic stability in sphere-based mode; however, as a statically stable ground vehicle this made it prone to dynamic instability. This was problematic for large accelerations, which caused Vertigo to lean or tip. When leaning, only one or two wheels contacted the ground and yaw was induced. Recovery from this often caused large overshoot, sometimes to the point of divergence or erratic behavior, depending on the controller. Frequently, these departures were caused by the paused Vicon connection. In this case, dynamic instability was merely the mechanism of failure, with poor connection as the cause.

Embedded Signal Processing

Unfortunately, Qwerk was programmed with no means of directly controlling the voltage to the motors. Control is prescribed with a scalar reference value that is sent to Qwerk. This value is between -50000 and 50000, and represents a unitless angular velocity factor. Qwerk uses this value as a final value in a trapezoidal trajectory generator. It then implements a PID controller with EMF feedback to follow this velocity profile. This entire process is embedded, and the original trajectory generator and PID gains had an extremely slow response; it took roughly 10 seconds to reach full speed from rest. This is two or three orders of magnitude too slow when dealing with a system that samples around 25 Hz. Mending this problem was discussed with the communication architecture, but the issue was mentioned here because it was discovered through implementation, and delayed progress by several months.

Yaw Control

If yaw angle was not controlled during tracking, the robot was free to rotate under external influences. This meant forfeiting control of camera direction, and was especially problematic when controlling from the inertial frame. Any misalignment (from yaw rotation) with the two frames skewed all control application. This was only a problem for ground-based control, as most sphere-based methods only used Euler angles. Eventually, when translation is attempted, this will also become an issue for sphere-based control. To address this problem, two strategies for proportional yaw control were implemented.

The first method was accomplished by fixing the yaw angle. In this situation, a proportional controller regulated the angle, most often driving it to zero. This way, Vertigo was only prescribed translational motion. Simple proportional control was found to be more than sufficient, because yaw requirements were not very demanding. For gain determination, it was increased until recovery from a $\frac{\pi}{2}$ step showed

overshoot. The gain just prior to overshoot was chosen.

The second control strategy was similar, but assigned a trajectory rather than a fixed angle. This most often took a form resembling a retrograde orbit in ground-based control. Before a yaw trajectory could be applied, control input had to be transformed from the inertial to the body frame. This was done by multiplying the positions with a transformation matrix that accounted for the yaw angle; therefore, position control input was made statically independent from yaw at the time of measurement. To perform the retrograde reference trajectory, a user defined angular yaw frequency was multiplied by time, and translation was assigned separately. By applying a proportional yaw controller, Vertigo was forced to follow the proposed trajectory.

Implementation of this strategy was difficult, because Vicon returned its angle measurements between π and $+\pi$. Adding π to the measurement signal fixed the problem for conservative trajectories, but beyond one rotation there was no distinction between 2π and 4π. Many commonly used rotation transformation methods, including MATLAB's mod command, did not fully address this issue. This is because they conditioned the measurement back to zero when it crossed 2π, and the error appeared to be 2π. Methods were also attempted that conditioned the trajectory by resetting it to 0 at 2π, rather than the other way around. These did not work either because the error became -2π and the robot quickly completed a full rotation in the opposite direction to pick up the trajectory again. Eventually, a measurement conditioning method was developed that allowed any number of rotations. This bit of coding is convoluted, as it works with three incrementing counters, the mod function and a time delay; however, it effectively shifts the measurements to a continuous scale.

Including yaw trajectories was not without drawback. Its problem was similar to the error from undesired yaw disturbances when controlled form the inertial frame. That is, as control was applied, the robot translated and rotated together, thus curv-

ing the translations. For slow trajectories, this effect was negligible because the sampling rate updated yaw measurements reasonably fast. As yaw rate increased, so did the error.

State Feedback

LQR is a full state feedback method; however, Vicon does not measure rates. Therefore, half of the ground-based states were not measured for feedback, and none were for sphere-based mode. Derivatives of the position had to be taken for rates; however, implementation was a discrete process, so numerical derivatives were taken as the difference in position from the current and previous time step divided by the sampling time between the two. In addition, controllers had to be implemented in the time domain. This was challenging because many continuous and discrete methods for controller design are not carried out in the time domain, accordingly, either extra measures had to be taken with their application, or they were deemed too disparate to be applicable.

7.3 Ground-Based Implementation

Implementation began with the ground-based mode, because it was a safer and more attainable starting point than sphere-based control. The first code to successfully make the link between Vicon and MATLAB tested the position and attitude of the Vicon object. Using the sim command, MATLAB extracted the Vicon information through a manufacturer provided Simulink plant that was able to record measurements. This program was useful throughout the project for checking the coordinates of the robot, and analyzing how well the object was defined by Vicon.

Next, a program was built to complete the full communication loop by sending

commands to the robot, and using the Vicon information as feedback. This controller tracked a one-dimensional user defined step function with if-statements and constant magnitude control inputs. Broken down, the logic followed that Vertigo should be moved forward if the difference between desired and measures position was negative, and backward if positive. This program helped to start piece together the coordinate and actuation sign convention for the loop. For programs that introduced new functions, if-statements were used rather than proportional control because they allowed exact specification of input, preventing motor saturation.

Building off this, a two-dimensional proportional controller that tracked a step function in both x and y was constructed. By converting the x if-control into a proportional gain that was multiplied by the position error, and then mirroring application to the y axis, this controller verified that Vertigo has decoupled omni-directional motion on the ground. It was also used to confirm the inertial axis convention, establish the safest way to terminate control, and experiment with control input and gain values to determine reasonable ranges. Running these experiments began to expose some of the flaws in using the Vicon system. Nonspecific object definitions meant that the center of the Vicon model did not exactly match that of the actual robot. Also concerns about yaw control were seeded here, as minor but erratic rotational drifting was observed for large step functions. Figure 7.1 shows how this controller tracked a two-dimensional step from (0,0) m to (0.5,0.5) m. Plotted with time as the z-axis, the proportional control is apparent as velocity is greater at the beginning, and consistently decreases as the error reduces. Plots of the control input showed the inverse of this trend as magnitude diminished with time.

To kick off yaw control, and establish the Vicon defined convention, if-logic was employed again. Once directionality was confirmed, proportional control was introduced. It was discovered that the gain and resultant control magnitudes were much

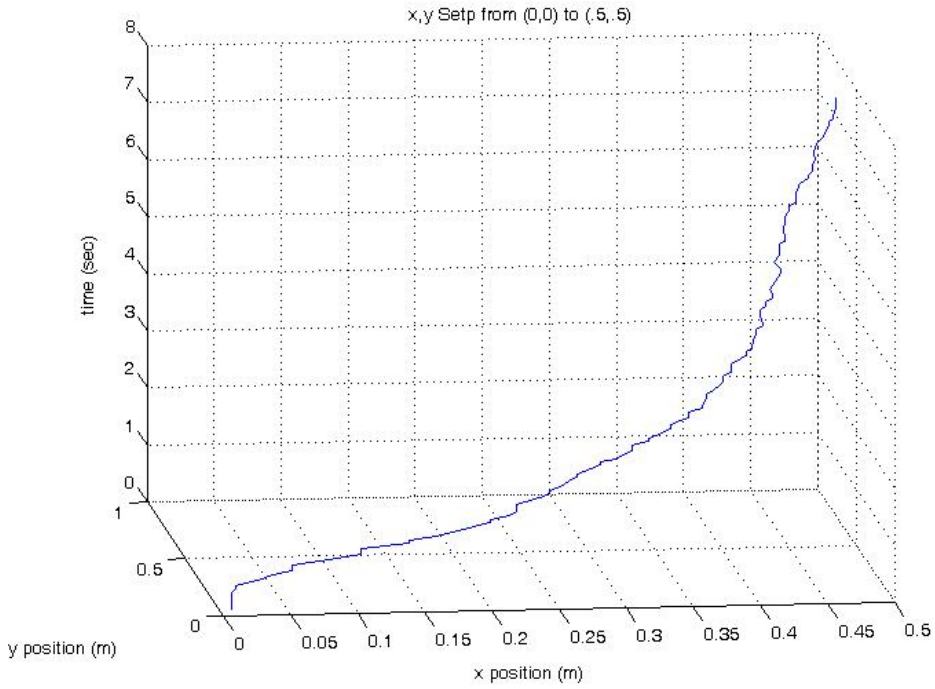


Figure 7.1: Proportional two-dimensional step track.

more conservative for yaw than translation. This was because the error to recover from was limited to $\pm\pi$, and control was cooperatively applied by all four motors.

One of the goals of implementation was to compare the simulated LQR controller with the physical response. This simulation was discussed in the Control chapter, and was specified to track a standardized circular trajectory with a 1 m diameter, traversed at 0.3 rad/sec. To begin work on this, the proportional controller was applied to both directions, and coded with a variable reference point. By specifying how the reference should change with time, this was effectively modified to track a trajectory, rather than steps. For generating the trajectory, the tic toc MATLAB function was used to synchronize the code to real time. The trajectory was specified in the same way as the simulation, with sine and cosine waves for the two axes. Variations of these allowed easy specification of sinusoidal and elliptical trajectories as well. The

proportional gain was chosen empirically; the results included in Figure 7.2 represent the most successful gain found. With norm-error=2.3897, this controller fell short of the norm-error=1 criteria, but was respectable for such a simplistic form of control.

To work out how time derivatives would be taken, a PID controller was implemented next. Numerical derivatives and integrations were used. This required that a delay of one time step be induced to calculate the first time difference. The Ziegler-Nichols method was used to specify a starting point for the PID gains; they were experimentally tuned from there. Figure 7.3 shows that increasing the complexity of the controller to PID reduced the norm-error by 0.5 compared to the proportional controller.

LQR can be thought of as a form of PD control, which is similar to both P and PID control; however, LQR's primary advantage is that it is a method for determining the optimal gains, so tuning is ideally not necessary. Gains from the simulated LQR controller were used here to maintain consistency. A progressive series of LQR controllers were written to manage various aspects of motion. The response for tracking the standardized trajectory and the associated norm-errors are included for comparison with other controllers and the simulated results.

The first LQR controller was implemented similarly to the PID controller, with the exception that it converted the input from torque (Nm) to angular rate (rad./sec.), and then scaled it to a corresponding Qwerk input. This allowed mathematically calculated gains to be written directly into the code, and converted internally. Observing the performance of this controller confirmed the need for controlling yaw. Therefore, another controller was created that expanded LQR to include proportional yaw control. This was applied using control superposition (see Appendix A) in moderation to gently hold the angle.

Figures 7.4 and 7.5 illustrate the importance of yaw control for this system. The

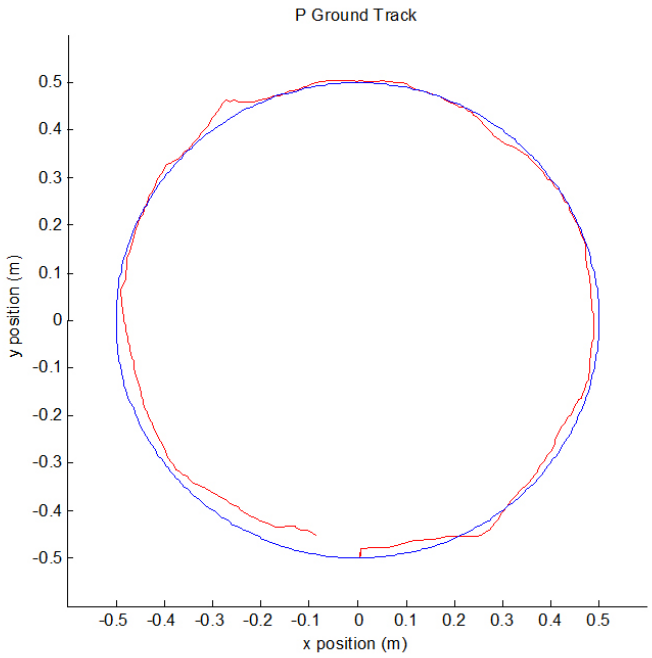


Figure 7.2: Circle trajectory tracking using proportional controller with proportional yaw control: norm-error=2.3897.

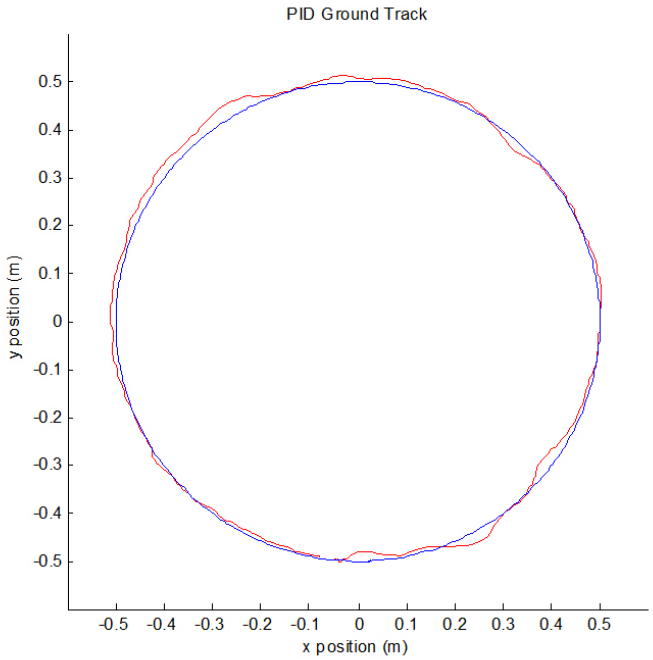


Figure 7.3: Circle trajectory tracking using PID controller with proportional yaw control: norm-error=1.8976.

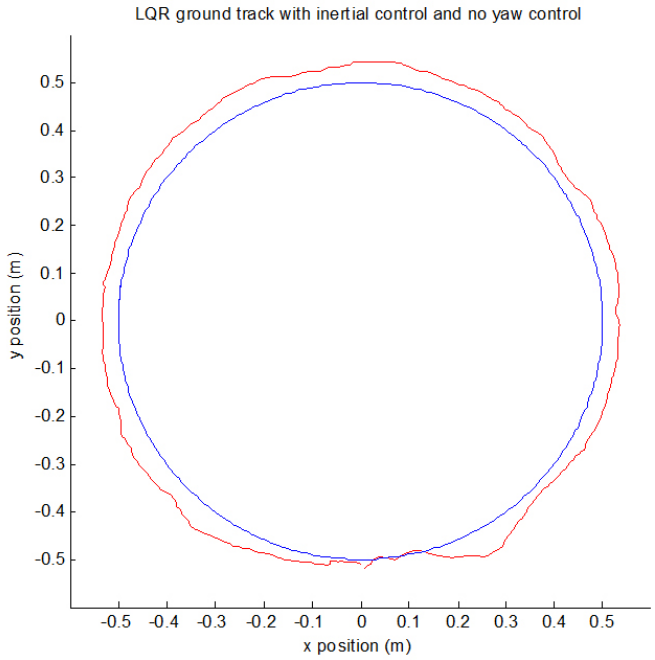


Figure 7.4: Circle trajectory tracking without yaw control: norm-error=0.8614.

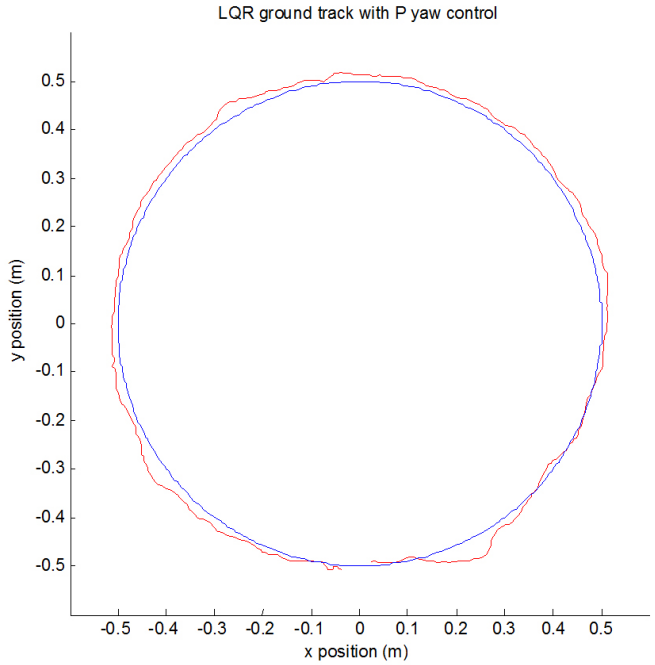


Figure 7.5: Circle trajectory tracking with proportional yaw control: norm-error=0.3866.

track started at the bottom of the circle and progressed counterclockwise. Shortly after starting, both experiments encountered an uneven section of floor which caused slip in the y direction. In Figure 7.4, where no yaw control was applied, this slip and recovery undesirably induced a slight yaw rotation. Because control was being applied from the inertial frame, this rotation skewed the control input for the remainder of the trajectory track. In Figure 7.5, the same slip occurred, but the fixed yaw control accounted for any undesired yaw that may have been induced, and the track was quickly resumed. The yaw controlled response had a norm-error about 2.5 times smaller than the uncontrolled response. These results capture just one example; the uncontrolled response experienced slightly different yaw perturbations each experiment, and its accuracy varied accordingly. If the accumulated disturbances ever exceeded $\pm\frac{\pi}{2}$, complete divergence occurred because control was applied along the wrong axis.

To demonstrate some of the disturbances, Figure 7.6 shown the test trajectory being traversed 4 times while holding yaw to zero with a proportional controller. By completing the circle multiple times, it is clear where the track consistently departed from the trajectory. These represent uneven ground that caused slip. The lower, lower-left and right portions of the circle show examples of this. There are also two pronounced departures near the top, these were caused by temporary communication failure between Vicon and Simulink. While delayed, control was held constant, resulting in the signature tangential wander. The departure to the right happened on an even part of the floor with surefooted actuation, so the recovery was accordingly swift. The tangential departure to the left was on uneven ground that was prone to slip, as seen by the other three tracks; this resulted in a slightly longer recovery time. The only other common obstacle that is not seen here is a Vicon object tracking failure. This happens when Vicon loses sight of one or more markers and does not

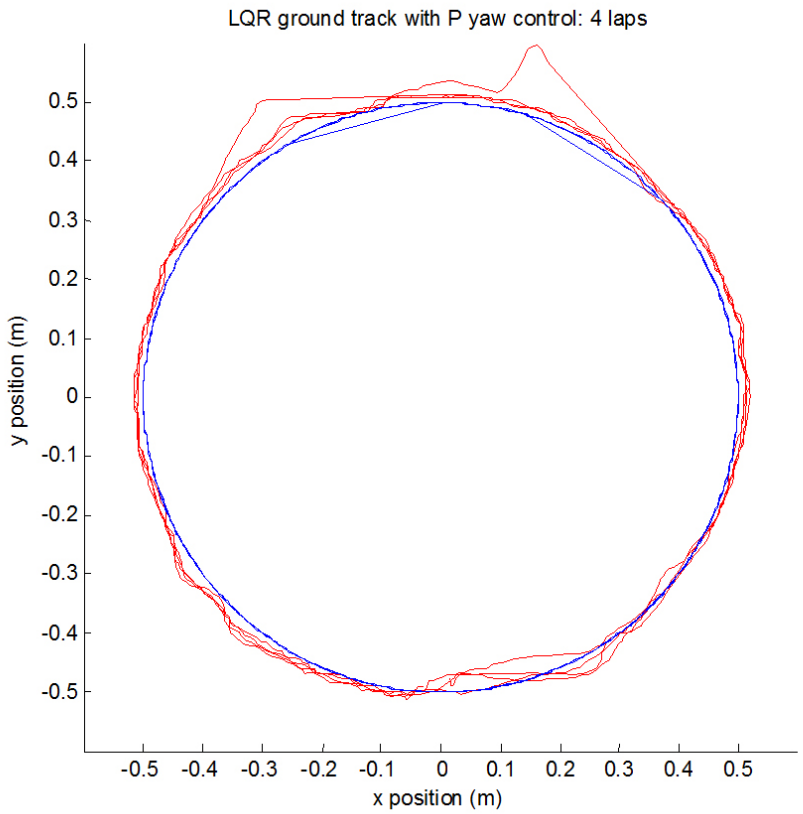


Figure 7.6: 4 circle trajectory tracking laps with proportional yaw control.

have enough information to track Vertigo. Vicon then sets the object's position to the origin and a signature spike will be seen in the track plot that goes to the origin and then back out to Vertigo's location once it is observed again. This results in an inaccurately high position error that cause the controller to send large inputs and huge departures occur. The frequency of this problem was increased for balancing controllers because changing roll and pitch angles were more prone to hiding markers.

One final modification was made to the ground-based LQR controller that helped to prove the concept of omni-directionality with yaw. This was done by transforming the control signal from the inertial frame, into the body frame. This allowed independent yaw and position trajectories to be specified. It also meant that yaw

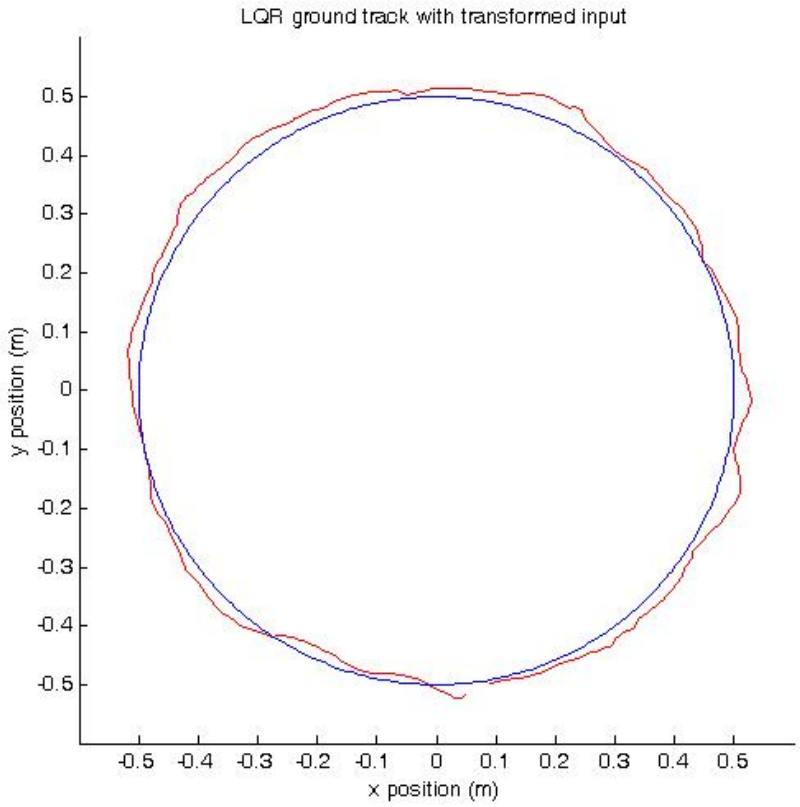
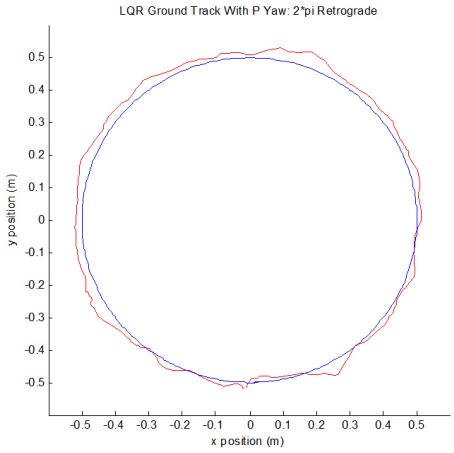
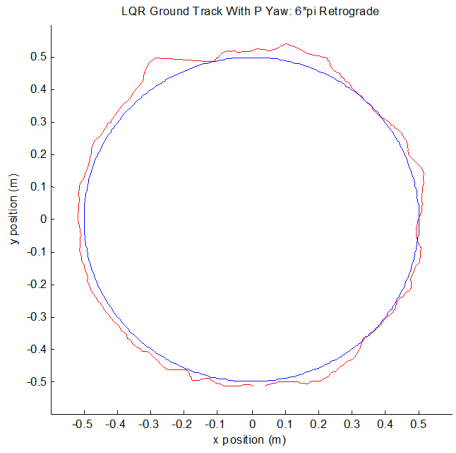


Figure 7.7: Circle trajectory tracking with transformed control: norm-error=0.5402.

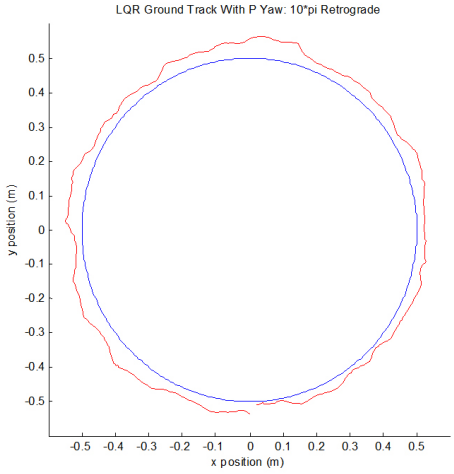
control could be omitted, and inertial trajectories could be tracked accurately in the body frame, despite yaw disturbances. Figure 7.7 shows the standardized trajectory being traced with transformed input. With norm-error = 0.5402, it outperformed the uncontrolled track; however, it could not quite match the response of controlled yaw with untransformed input. This is because actively controlling yaw increased the control when disturbances were encountered, which helped conquer them faster. With uncontrolled yaw, no additional input was sent to aid the LQR, and the response suffered slightly. Experiments were run with transformed input and fixed yaw control; however, they did not improve upon the yaw control significantly.



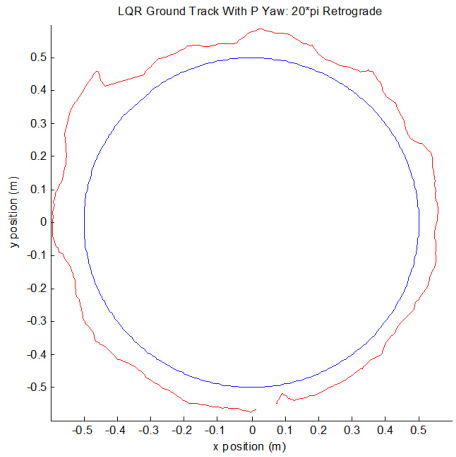
(a) Circle trajectory tracking with proportional yaw control for 2π retrograde: norm-error=0.5486.



(b) Circle trajectory tracking with proportional yaw control for 6π retrograde: norm-error=0.5805.



(c) Circle trajectory tracking with proportional yaw control for 10π retrograde: norm-error=0.7973.

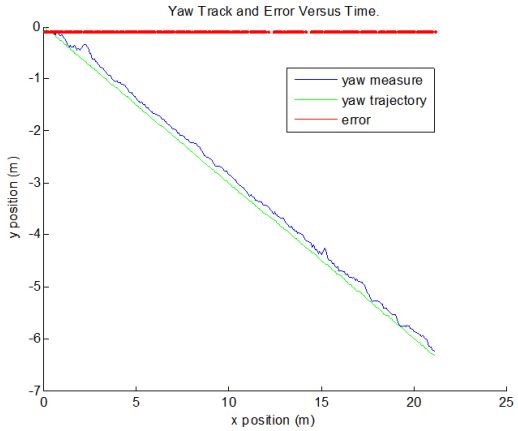


(d) Circle trajectory tracking with proportional yaw control for 20π retrograde: norm-error=1.3933.

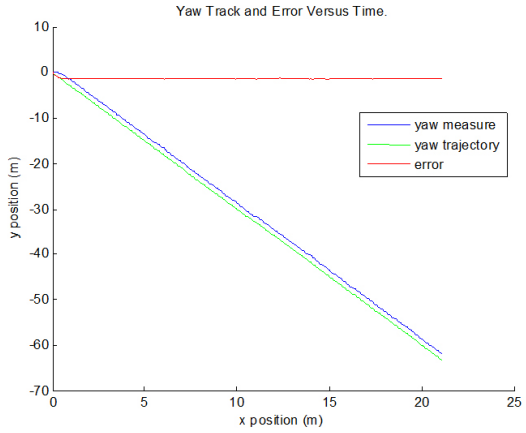
Figure 7.8: Circle reference trajectory tracking while using proportional yaw control for various retrogrades.

To analyze how prescribing both trajectories affected performance, experiments were run on the standardized test trajectory with an increasing number of retrograde yaw rotations. Figure 7.8 demonstrates how the norm-error increased with the number of retrograde rotations. As mentioned earlier, this is because control is held constant between sampling times. As the body rotates, directional translation skews, and a straight trajectory in the body frame becomes a curved path in the inertial frame. The greater this curvature becomes, the further out the track will move radially for retrograde rotation. If the yaw trajectory were in the other direction, the track plot would drift inward as the number of rotations increased. The two conservative trajectories did not yield much more error than the yaw fixed case, but in Figure 7.8(c) and 7.8(d), where the rotations were much faster, there was appreciable error.

The plots of yaw measurement and trajectory for the smallest and largest rotations in Figure 7.9 show that the method devised for conditioning the measurements works for any number of yaw rotations. These plots also include the errors that indicate how far off the trajectory the track is. Errors plateau at a small nonzero number because there must always be some error for the proportional controller to produce input signal. It is worth noting here that simply looking at how well the track fits the trajectory is not sufficient enough information to make discerned conclusions. As mentioned before, the time response is also important, so the norm-error values are an accurate basis for comparison because they take time into account as well as position errors.



(a) Yaw control during 2π retrograde.



(b) Yaw control during 20π retrograde.

Figure 7.9: Yaw control for 2π retrograde and 20π retrograde.

7.4 Balancing Implementation

Working towards the goal of building a preliminary foundation to expedite future work in balancing on the sphere, the next step was to conduct simplified balancing experiments. As with ground-based control, the first balancing controller exploited the safety and predictability of if-statements to establish the coordinate, error and control signal convention. This if-control was designed to balance an 8 foot long 2x12 inch plank that pivoted at the center. By translating along the top of the plank,

Vertigo was able to effectively control the pitch of the board. If the pitch error was negative, Vertigo would move forward, and vice versa.

This program was a valuable step because it exposed the need to keep all forms of motion in check, not just for performance, but safety. As the robot moved about, small disturbances in y and yaw accumulated, eventually causing Vertigo to fall off the board. This was successfully addressed by adding y and yaw components to the if-control. The only thing preventing this controller from balancing the board indefinitely was battery life, and Vicon connection lapses. These lapses caused Vertigo to quickly run off the board before there was time to catch it. More protection was clearly needed for balancing experimentation to safely continue; therefore, a 6 inch lawnmower tire inner tube was stretched around the middle of Vertigo. Partially inflated, this served well as a durable bumper for damping crash impacts.

Controlling board pitch became slightly more intelligent by converting to proportional control for x, y and yaw. The gains and magnitudes discovered in the ground-based proportional control experiments accelerated the process of finding appropriate gains here. The proportional controller easily balanced the board; however, the response was choppy, and certain gain values induced oscillatory overshooting that resonated, causing the board to sway. This was not an issue for if-control because its response was not consistent enough to resonate. Figure 7.10 shows the board starting with one end on the ground and plots the response as Vertigo brings it to balance. At 3 seconds, the board tipped to the point where it almost made contact with the ground. It recovered, but oscillation persisted.

To attenuate oscillation, PID control was called upon. Inclusion of the derivative component was especially beneficial, as it helped to damp the response. Figure 7.11 shows the significant improvement over proportional control. Around 5 seconds into the experiment, Vertigo raised the board to the balancing point with very little over-

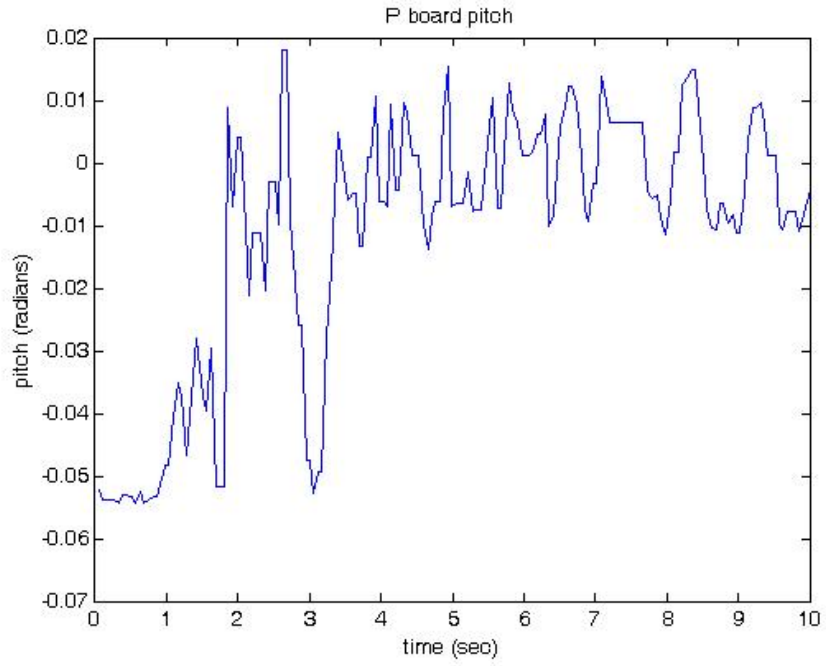


Figure 7.10: Balancing plank pitch with proportional control.

shoot. This plot also demonstrates how much noise is added by the Vicon measurements. For this experiment, the plank was fitted with markers, and its angle used for feedback to the controller. Therefore, any variation in the response in the first 4 seconds can be attributed to Vicon noise because the plank was resting on the ground at a fixed angle. Though this noise is undesirable, tracking the board was a significant improvement over the response when Vertigo was the Vicon object. Figure 7.12 shows how noisy the response was when Vertigo's angles were tracked by the same PID controller. This additional noise is partially due to uneven terrain, but the dominant contributor is actuation. Large gaps in the omni-wheels gave Vertigo a bumpy ride, so the measured angle at a given time did not represent net pitch well. This noise was so dramatic that performance was affected by it.

Tracking the board was just a temporary solution to circumvent the true prob-

lem: Vertigo needed better omni-wheels. This raised concerns about using Vicon as feedback for balancing on the sphere because tracking a different object would not be an option, and the improved wheel design would be extremely expensive and time consuming to manufacture.

Balance in 2-dimensions began by applying the if-controller for the board to both axes. The primary objective of this was to confirm the convention of the Vicon roll angle. This simple controller was able to balance a flat plate resting on a sphere. This was analogous to balancing the plank in 2-dimensions. The biggest challenge in this was recovery from large disturbances, which is best described with an example. If the pitch angle deviated to the point where the front edge of the board was on the ground, roll angle was held constant despite the efforts of y control. Therefore, Vertigo moved in the y direction to try to account for the small angle of the uneven ground. By the time x control brought pitch back to balance, y had veered far enough to deflect the roll angle to rest on the adjacent edge. This was not a problem when starting from a balanced point and the objective of establishing convention was met; therefore, further development was not necessary.

At this point, an informative collection of controllers had been created to prepare for balancing on the sphere. Although it surpassed the objectives of this project, early stages of sphere-based implementation were attempted. Building off the findings of previous controllers and experiments made it easier to understand the origins and severity of problems encountered in this. Proportional control was first attempted with the sphere. With Vertigo's angles as Vicon feedback, this controller applied input in the correct direction for balance, but was insufficient. Gains that were too low did not apply enough control to recover from any appreciable disturbance, but gains that were large enough to bring Vertigo back to the unstable equilibrium would overshoot to an unrecoverable angle. However, this controller was beneficial because

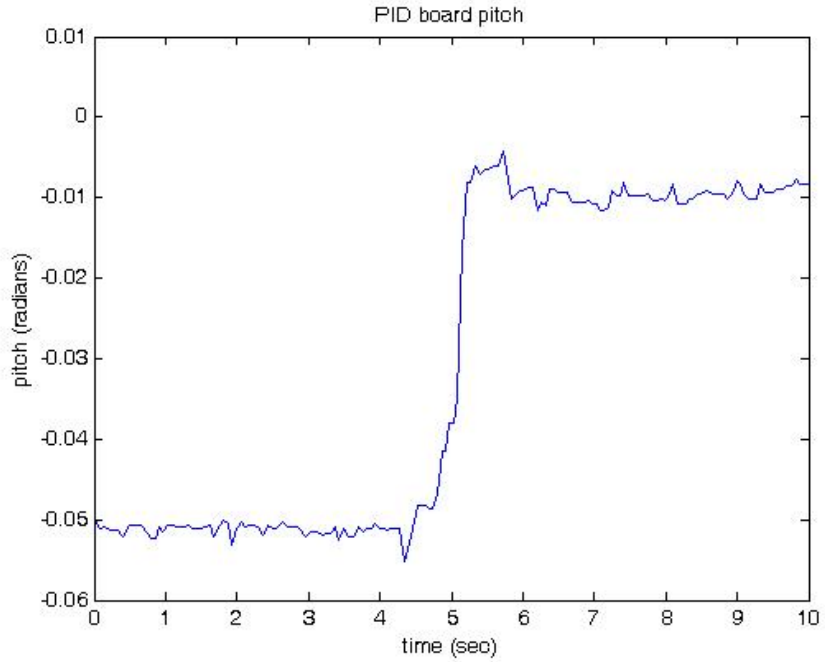


Figure 7.11: Balancing plank pitch with PID control.

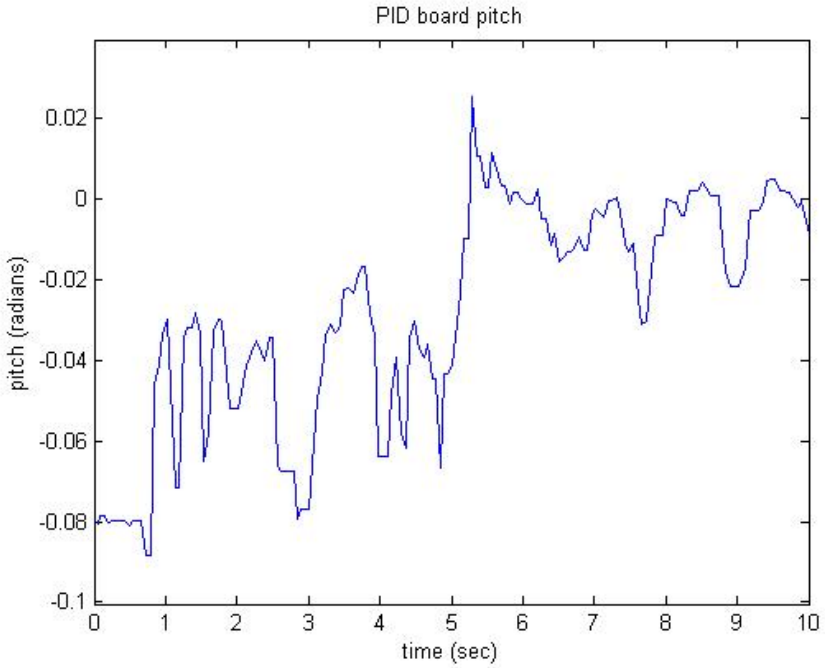


Figure 7.12: Balancing plank pitch with PID control by tracking Vertigo.

it started to demonstrate the specific challenges of balancing on a sphere.

PID control was then implemented in a more sophisticated attempt at balance. This gave better results; however, it could not sustain balance, only keeping Vertigo on the sphere for about two seconds. During this time, oscillations would diverge to an unrecoverable angle. Tweaking the gains to reduce overshoot limited the recovery angle such that it was not able to cope with the small deviations from noisy measurements and external disturbances. Since the damping component of the PID controller was most beneficial for balance, and tuning was difficult, LQR was attempted next in hopes that its optimal gains would solve the problem.

For LQR control, many techniques were tried for implementation and defining the Vicon object, but none were successful. LQR was appealing because it included the damping aspect of the PID control, and it gave optimal gains for the mathematical model; however, it is possible that model errors were sufficient enough to invalidate the results for the physical system. Also, the framework of implementing LQR meant that measurements had to be related to the mathematical states of the system. This used a geometric measurement model that relied on how accurately the Vicon object was defined to represent the physical system. Errors from this were apparent because the performance of the controller quickly worsened as balance was attempted further from the origin where the uncertainties were magnified.

Based on the success of the estimation simulations, the continuous-discrete extended Kalman filter seemed well suited to accompany LQR control and solve the problems with noise, modeling error and measurement error. However, three main issues prevented this from being successful. The first was with initialization. Manually placing Vertigo close enough to the filters initial conditions was difficult, and often caused immediate failure. Also modifying the filter to account for larger initialization errors diminished its effectiveness. The second problem was that Vicon

did not always define the object well enough. This had similar repercussions to the initialization error. The third and most influential problem was the inconsistent loop sampling time as signals navigated the tortuous communication architecture. EKF requires knowledge of the sampling time for tuning, and to propagate a priori estimates for the next time step. When sampling time varies, propagations are erroneous, and the filter tuning is compromised. Despite the shortcomings, these attempts were not in vain. EKF is extremely powerful, and it is likely that continued work will enable it to provide state estimates reliable enough for Vertigo to balance on the sphere.

Chapter 8

Conclusions and Future Work

This thesis introduced an autonomous biaxial robotic inverted pendulum as a new experimental platform for research in controls and dynamics. Designed to balance and navigate on a sphere, it is theoretically capable of exceptionally agile motion. This, along with its reconfigurability, enables it to model a wide range of other, more complex systems. Its unique abilities also permit original work in rover locomotion, and as an educational tool. Covered in this thesis were the design, mathematical modeling and analysis, communication architecture, control, estimation and early implementation.

As an original platform, proper hardware design was a major effort. The original objective was to create a simplistic and robust testbed with minimal restrictions on its motion. The first prototype, Vertigo 1.0, balanced on a sphere, and had omni-directional motion. Actuation methods were the cornerstone for enabling its dynamics. A second design iteration, Vertigo 2.1, was then developed with new actuation that offered the addition of yaw control to the existing omni-directionality and biaxial inverted pendulum dynamics. Both of these generations required only a handful of moving parts, and were successful in protecting onboard components from impact.

However, working with them gave rise to ideas for improvement. Results from control implementation showed that a need for new omni-wheels was the most urgent of these improvements. The gaps between the rollers caused chattery motion, which affected measurement quality. A new design for near seamless omni-wheels was generated, but cost and time of manufacturing made them impractical to this point. In the future, it would behoove the project to incorporate these new wheels.

Also proposed is a design iteration that fits Vertigo with only three legs. This would make the Vertigo 2.1 drive equal-actuated while preserving all modes of motion, and making more consistent contact with the control surface. The only drawback is a sacrifice of independent actuation; however, motion would remain decoupled, so control could be decomposed and applied in representative components relatively easily.

The mathematical modeling of Vertigo proved valuable for its development. Its accuracy was confirmed through the response of simulations and agreement of analysis results with theory. Analysis of the equations focused on observability and controllability. These results were useful in confirming the validity of control and estimation methods that would be applied. By looking into the inverse condition number of the controllability and observability matrices, insight was gained as to how states influenced their condition. This also guided the hardware assembly configuration to help achieve desired performance. Future work on mathematical modeling should focus on further improving accuracy in matching the physical system, as this is a suspect for implementation difficulty. Deriving the equations with various methods and assumptions should give an idea of how to better represent the dynamics. Also, system identification techniques could be employed to extract parameters from the physical response. These efforts would almost certainly enhance implementation success and the validity of simulations.

Establishing the communication architecture was a laborious process, requiring the use of five different coding languages. Eventually, an external sensing loop was closed where Vicon measurements were sent to Simulink, and then passed on to MATLAB to be conditioned for feedback in to a control loop. The output of this loop was sent to Qwerk through a Java based wireless connection. Qwerk received this control signal as input to its own PID control loop that tracked a velocity trajectory. Output from this embedded controller was sent to the motors, and their back-EMF measurements were fed back to the PID loop as they carried out the control. In the future, this network will be greatly simplified because a new version of Qwerk's firmware that supports the onboard sensing architecture will be released soon. Vertigo will no longer be dependent on Vicon measurements; a laptop and wireless router will be the only equipment needed. This means easy transportation and set up almost anywhere, and Vertigo can be made fully autonomous. The new firmware also promises to support passing of the embedded PID gains and trajectory parameters. This will enable further tuning for improved response, and back-EMF measurements can be added with low weighting to MATLAB feedback loops.

Development of control methods spanned a wide range of motion. Starting with the ground-based configuration, a standardized test trajectory was developed to compare the results of simulated and implemented experiments. The majority of the methods simulated in this thesis were optimal control techniques. These gave ideal gains for respective specified cost functions, and removed some of the guesswork from tuning. Several sphere-based controllers were developed for balance and translational motion; simulation of these yielded exceptional results that confirmed Vertigo's conceptual principles. The most sophisticated and successful was an augmented LQR tracking controller that balanced and translated Vertigo according to a cost function that effectively minimized weightings of states, power and time.

Many other optimal control techniques were attempted for sphere-based control, but acceptable results were not attained. These included an analytical power-time optimal method, a power optimal shooting method for translation, and a gradient based method for translation. These techniques are highly sensitive to tuning and initial guesses, but are very close to converging properly; therefore, continued work on them will likely be fruitful. Success is especially hopeful for the gradient based and shooting methods, as they are complementary methods, and are both near completion. If either one of these methods is worked out, its results can be used as initial guesses for the other.

For estimation, the problem was put into the Kalman filter framework, and simulated. These results showed its success at providing estimates for unmeasured states while attenuating noisy signals. It was then shown that increasing the number of loop iterations between measurements updates improves the estimate further. To fully explore the impact of sampling times, the simulation was run at twenty five different combinations of measurement and output sampling times. The magnitudes of the settled 3σ bounds were plotted as a mesh surface which showed more measurements and loop iterations improved accuracy, but the affect was much more pronounced for measurement sampling. The EKF is not a true optimal filter because it applies a linear method to a system that is by definition nonlinear; however, it can still be used with exceptional results as demonstrated here. More work can be done to use this filter to estimate unknown parameters of the system. This is an extremely advantageous feature of the EKF because it adds functionality as a real time system identification technique to update the dynamic model and improve performance. For Vertigo, the unknown friction coefficients are likely to be the most beneficial parameter to estimate because they cannot be measured, and pertain to slip. Work on this has already begun, but revealed that defining new states that included an unknown parameter

make the system unobservable, and the process is not applicable. Continued research in this area is likely to uncover a method for circumventing this.

In accordance with the objectives of this research, the majority of future work will be in implementation. A large bank of ground-based and balancing controllers were successfully implemented to serve as a spring board for continued development of sphere-based implementation. With the current communication network, further work on the implemented EKF and LQR combination is most promising for balancing on the sphere. However, progress should flourish with the addition of improved omni-wheels, and the updated Qwerk firmware when it is released. The onboard sensing loop architecture should solve many of the timing and connection problems currently faced. Also, as the mathematical model is improved, implementing theoretical results will be more straightforward. With the accumulated success to this point and expected results of future work, the Vertigo platform promises a rich and prolific future.

Bibliography

[1] L.-H. Chang and A.-C. Lee, "Design of nonlinear controller for bi-axial inverted pendulum system," *IET control Theory Appl*, vol. 1, no. 4, pp. 979–986, 2007.

[2] S. G. Kobilarov, M.B., "Near time-optimal constrained trajectory planning on outdoor terrain," *IEEE International Conference on Robotics and Automation (ICRA2005)*, pp. 18–22, 2005.

[3] G. A. K. T. B. LAUWERS and R. L. HOLLIS., "A dynamically stable single-wheeled mobile robot with inverse mouse-ball drive," *IEEE International Conference on Robotics and Automation*, 2006.

[4] G. A. K. T. B. LAUWERS and R. L. HOLLIS., "One is enough!," *12th Intl Symp. on Robotics Research*, 2005.

[5] J. Dewey and P. Byerly, "Bulletin of the seismological society of america.," vol. 59, no. 1, pp. 183–227, 1969.

[6] S. N. X. Y. Hui-Di Zhang, S Hi-Rong Liu, "A neurodynamics based neuron-pid controller and its applications to inverted pendulum," *Third International Conference on Machine Learning and Cybernetics*, pp. 26–29, 2004.

[7] T. Uranaka, "Scientist claims he made segway predecessor in '86," *The Japan Times*, 2001.

[8] Y. K. Feng, Qing, "Design and simulation of a control system of an inverted pendulum," *Robotica*, vol. 6, no. 3, pp. 235–241, 1988.

[9] Y. Ha and S. Yuta, "Tracking control for navigation of self-contained mobile inverse pendulum," *Institute of Information Sciences and Electronics*, 2000.

[10] S. Y. Ryo NAKAJIMA, Takashi TSUBOUCHI and E. KOYANAGI, "A development of a new mechanism of an autonomous unicycle," *IEEE*, 1997.

[11] S. Inc., "www.segway.com," 2008.

[12] K. T. H. Arai and N. Shiroma, "Time-scaling control of an underactuated manipulator," *Journal of Robotic Systems*, vol. 15, pp. 525–536, Sep. 1998.

[13] A. Vera, "Human and robotic exploration," *NASA Ames Research Center*, Feb. 2007.

[14] Support, "Qwerk hardware guide," *Charmed Labs*, vol. 1, Sep. 2006.

[15] K. Corporation, "Transwheel powered bidirectional wheels, swivel wheels and swivel tables," 2008.

[16] J. Electronics, "Dc-motor specification sheet," Nov. 1999.

[17] Crossbow, "Imu user's manual modles: Imu300cc- imu400c- imu400cd-," *Document 7430-0003-03*, June 2005.

[18] Renco, "Hollow-shaft encoders specifications sheet," *RCH50*, 2005.

[19] T. T. R. Kit, "www.terk.ri.cmu.edu," *Carnegie Mellon University Robotics*, 2007.

[20] Logitech, "Webcam c-series specifications," *Communicate Overview*, 2009.

[21] P. P. Inc., "Technical product data sheets: Delrin," *www.plastic-products.com/spec*, 2009.

[22] Faulhaber, "Product specification tables," *micromotor*, Jan. 2008.

[23] Faulhaber, "Static-continuous callibration tables," *micromotor*, Jan. 2008.

[24] T. T. R. Kit, "Qwerkbot software api quick reference," pp. 1–6, 2006.

[25] K. Ogata, *Modern Control Engineering*. Prentice-Hall Inc.Addison-Wesley, 2007.

[26] C.-T. Chen, *Linear System Theory and Design*. Oxford University Press, 1999.

[27] C.-T. Chen, *Linear System Theory and Design: Transfer Function, State-Space, and Algebraic Methods*. Oxford University Press, 1993.

[28] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Transactions of the ASME–Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.

[29] J. L. Crassidis and J. L. Junkins, *OPTIMAL ESTIMATION OF DYNAMIC SYSTEMS*. CRC Press, 2004.

Appendix A

Supplementary Concepts

A.1 Introduction

The objective of this appendix is to provide cogent explanations for some of this platform's non-intuitive aspects. Many of the items addressed can be attributed to Vertigo's nonlinearity and unique design. Mathematical backing is provided accordingly throughout this thesis; however, this appendix focuses on clearly conveying the ideas in plain English. The topics of directionality, translational motion, actuation, control superposition, pivot drift, sphere properties and weight distribution are commonly misinterpreted by those first being introduced to this system. They are also critical to the proper understanding of how it works, and therefore warrant discussion.

A.2 Directionality

The directionality of a system refers to its mobility and is often mislabeled in robotics when describing motion. Categories for directionality are loosely defined, and many

vehicles do not fit nicely into a particular profile. This section provides simple definitions and explanations for common types of directionality, and establishes the convention to be used throughout this thesis. Figure A.1 illustrates the progressively expanding mobility that climaxes at omni-directionality with yaw, which describes the motion of Vertigo.

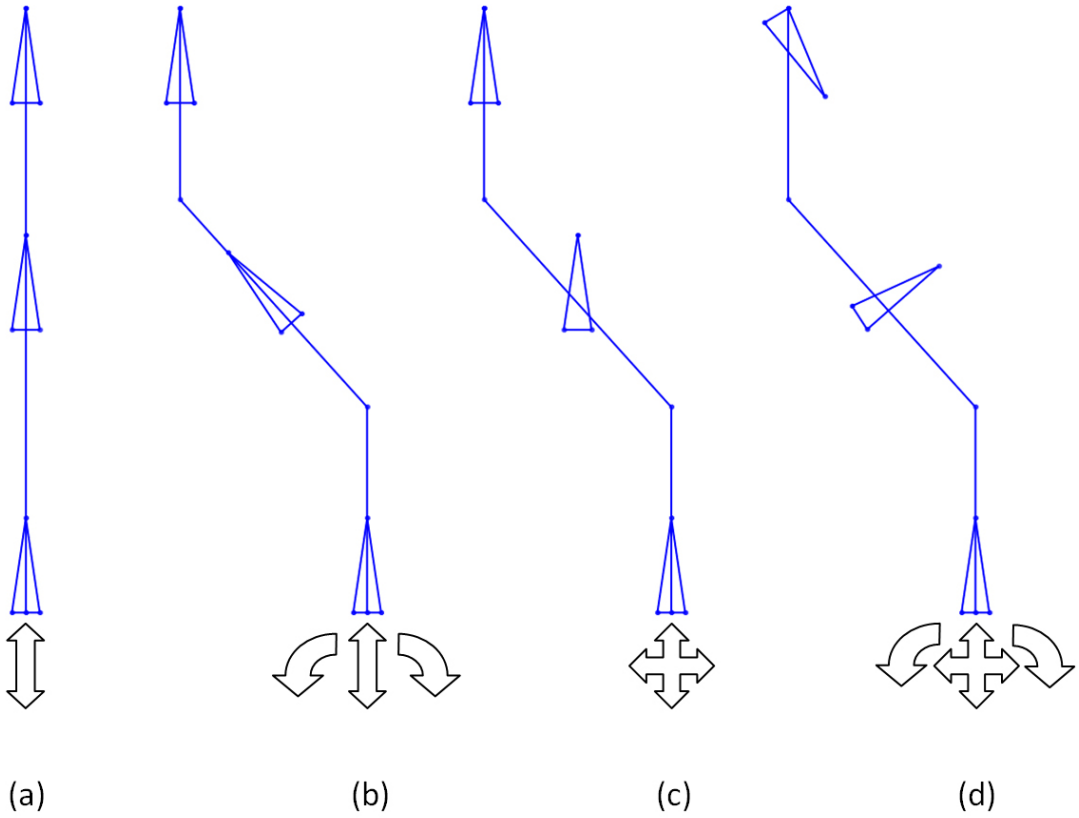


Figure A.1: Directionality (a) Bi-directionality (b) Bi-directionality with yaw (c) Omni-directionality (d) Omni-directionality with yaw.

Bi-directionality (Figure A.1(a)) simply means that an object is capable of moving forward or backward, along a single axis. A cart on a track is an example of this, where motion is constrained to one degree of freedom. Bi-directionality with yaw (Figure A.1(b)) is a more common form of motion. In this case, the vehicle has

the additional ability to rotate (yaw). Differential drive vehicles, such as tanks and wheelchairs, are examples of this. They often require only two driven wheels to jointly control their two degrees of freedom, and have additional passive wheels or casters to maintain their static stability. Nearly all mobile robots fall into these two categories.

Omni-directionality (Figure A.1(c)) is among the most commonly misused terms because it is considered a "holy grail" for robotic motion. It refers to a vehicles ability to move in any direction on a planar surface. This means that the vehicle does not need to be facing in a particular direction to carry out a maneuver, but also does not have the ability to intentionally control the direction it is facing. Controls in the x and y directions are usually decoupled for these two degree of freedom vehicles. This type of motion improves maneuverability and response time because the vehicle does not have to spend time "steering" to avoid obstacles.

Omni-directionality with yaw (Figure A.1(d)) is the most ambitious form of planar motion, having all three possible degrees of freedom. It means that the vehicle is able to move in any translational direction on a surface, and control its yaw angle independently. The primary advantage of this over omni-directionality is that yaw can intentionally be controlled at all times, so sensing equipment can always be facing its subject. Both ground-based and sphere-based Vertigo have omni-directionality with yaw.

A.3 Translational Motion

The method by which Vertigo is able to navigate from point to point while balancing is often misunderstood. The key to this is controlling the point of contact with the ground with respect to the center of mass. In this way, it is very similar to the way

a human moves, and when explained step by step is rather intuitive. Figure A.2 illustrates the following description as Vertigo translates from point A to point B and time passes.

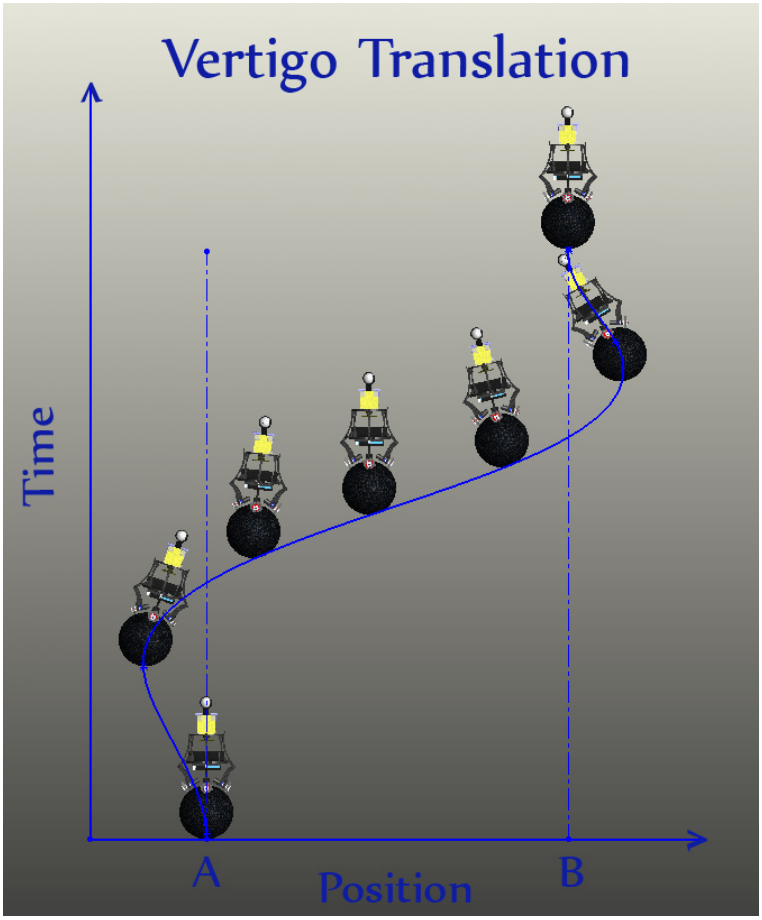


Figure A.2: Stages of translational motion.

Translational motion of Vertigo, or any inverted pendulum, can be broken down into seven distinct stages. The first stage is simply the initial condition, where Vertigo is vertical and stationary at point A. Now, to move forward, the robot must match its acceleration with the angle of lean. Accordingly, lean must first be initiated in the second stage by shifting the point of contact with the ground backward. This stage is frequently confused despite humans subconsciously doing the same thing before

walking by rolling weight back onto the heel. The third stage is acceleration. This stage is essentially a recovery from the initial lean in the previous stage, and the greater the angle of lean the greater the acceleration. This is why sprinters start a race on all fours; as soon as their hands are lifted they are leaning at an extremely large angle and can impart greater force to accelerate than someone starting upright.

Stage four is cruise. Here, the robot is moving at a constant velocity and is perfectly vertical (assuming no friction or drag). Everything is mirrored about stage four, so the deceleration in stage five is carried out similarly to acceleration, except the lean is in the opposite direction and the brakes are applied. Stage six acts to terminate the lean. This can be carried out in two slightly different ways; the first is by applying the brakes perfectly so that Vertigo pivots back to vertical with a velocity of zero and exactly at the desired destination. The second is the opposite to lean initiation in stage two, the final destination is overshot slightly as the motion is slowed, then the recovery acts to resume vertical and find the final destination. The second way is more time efficient because the first theoretically takes an infinite amount of time; however, both are actually different manifestations of the same process. Stage seven is the final state of the robot, where point B is reached, and all translation has ceased.

A.4 Actuation

The method used here for actuating the angles of an inverted pendulum is unique to Vertigo. When balancing an inverted pendulum, the goal is to control the base, in this case the sphere. What sets Vertigo apart is its actuator orientation and use of omni-directional wheels that allow control to be applied to the same surface in various directions without interference. Four spherically orthogonal motors allow almost any

size sphere (restricted only to a minimum size) to be used as a base for Vertigo. Taking this to the extreme results in a sphere of infinite radius which is a flat surface; this is why the ground-based configuration of Vertigo is possible. The omni-directionality comes from the two inch diameter wheels that are inlaid with rollers around their circumference. This configuration allows control in the direction of rotation of the motors, and slip perpendicular to this which decouples the directional control.

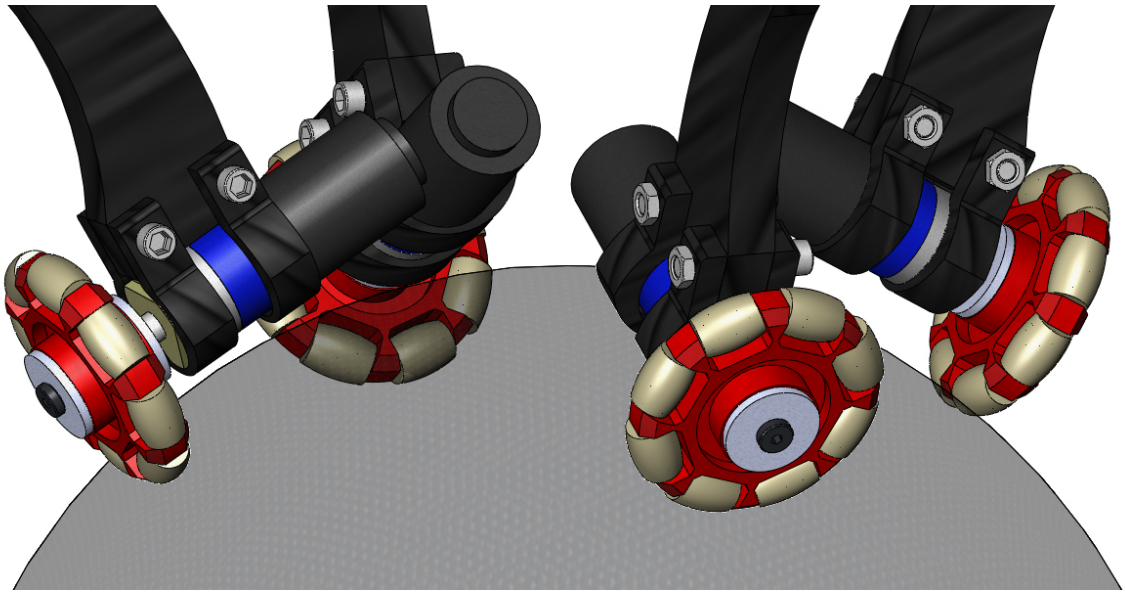


Figure A.3: Spherically orthogonal omni-directional actuation.

Here, directional control is referring to the motion of the sphere's surface with respect to the robot's body. "Controlling direction" is somewhat specious wording; really, the motors are applying a torque which acts to rotate the ball. This rotation causes Vertigo to lean (as mentioned in stage two in the previous section), and is how the angles are controlled. By calling upon opposing pairs of motors, roll and pitch are decoupled and independent. To control yaw, all four motors rotate in the same direction. If the friction between the ground and the ball is sufficient, the robot will rotate while the sphere remains stationary.

A.5 Control Superposition

Previously, it was shown that Vertigo's directional control is completely decoupled through actuation of opposing pairs of motors, but its yaw control requires the cooperation of all four motors. For the continuous model, the solution to this can be handled by superposition. The same principles apply for both the sphere and ground-based configurations of Vertigo, so only ground-based control is considered here because it is easier to visualize.

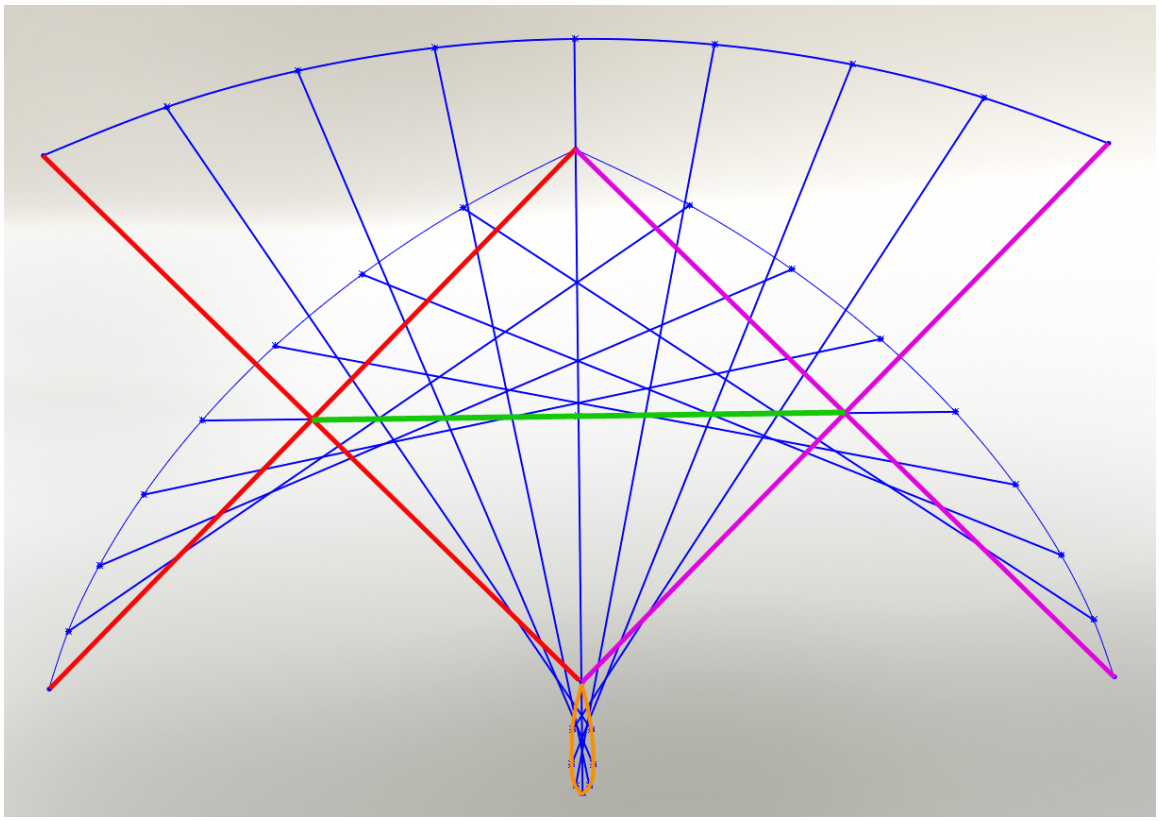


Figure A.4: Ground-based trace of translation with rotation.

Figure A.4 shows the trace as ground-based Vertigo travels along the straight

green trajectory and executes a $\frac{\pi}{2}$ yaw rotation with constant angular velocity. The red X represents a simplified top view of Vertigo at its initial position. As its center of mass progresses along the straight green line, and yaws at a constant rate, periodic snapshots are taken and traces of the four corners of the X (the motor locations) are drawn. These traces represent the paths that each individual actuator takes. Since the actuators only have to control motion perpendicular to their axle, trajectory projections can be drawn onto the plane of the wheel for each respective actuator trace. In this way, each actuator need only control the projected motion. For continuous control, this is equivalent to superposing the control. More specifically, the control for translation (transformed into the body frame) and yaw rotation can be determined separately, then added together to compose a complete control profile for the maneuver. To transform the translation control into the body frame, information about the yaw angle is needed. In practice, this is a problem because continuous knowledge of yaw and continuously transforming control into the body frame are both impossible. The reason behind such difficulties is discretization. Yaw information comes from sensors that have a discrete sampling time and transformation takes place in coded loops which do not provide continuous output. This naturally brings up the discussion of the discrete-time controlled system.

In the discrete-time control system, yaw is measured, control is calculated and transformed, and voltage is applied to the motors, at every time step. Between steps, the voltage to the motors is held constant. To demonstrate the problem, consider ground-based Vertigo under discrete-time control with a rather large sampling time of one second. Now suppose that translational control for forward motion is determined, and yaw control was separately determined to follow a rotational trajectory. Now, when those two inputs are superimposed and applied as voltages to the motors, that signal will remain constant throughout the time step. Therefore, at the start of

the step, the state for which control was calculated, the input will be appropriate; however, at the end of the time step, one second later, the straight line control will have curved as a function of the yaw rotation. For the second time step, the translation control will try to bring the robot back to the trajectory, but again yaw rotation will cause divergence and prevent the desired control from being carried out. The end result for this scenario would be a track which resembles a wavy bias error.

The problem of translational control divergence is a function of the yaw rotational velocity, and the sampling time of the system. In practice, fast sampling times and conservative yaw trajectories help to mitigate these errors. For most practical applications, a yaw trajectory can be determined that allows superposition to be taken advantage of with negligible error. If more evasive maneuvers are required, empirical knowledge of the predicted error can be applied to help compensate. This error can also be determined mathematically; however, constant sampling time must be assumed and this is not the case with Vertigo's current control loop.

A.6 Pivot Drift

A famous benchmark problem in control theory is the inverted pendulum on a cart, where the angle and position of the pendulum are controlled with an input force applied to the cart, as seen in Figure A.5. In many ways, the dynamics of this system are conceptually similar to those of Vertigo; both are inverted pendulums, and both apply an input force (or torque) that controls the base. However, the feature that differentiates these two problems is the sphere that Vertigo balances on. Balancing on the sphere has two dominant affects on the dynamics that prevent the control methodology from being used interchangeably between these two problems; the first

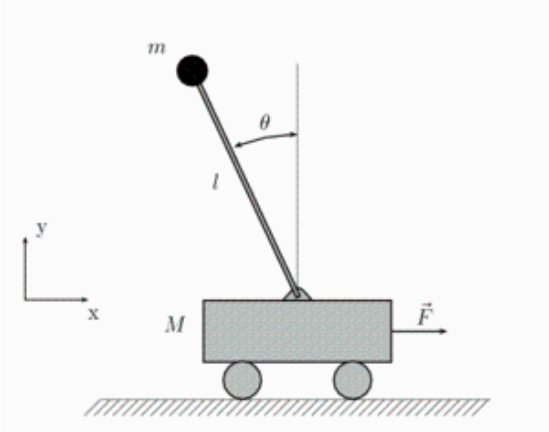


Figure A.5: Inverted pendulum on cart.

is pivot drift, and the second is the rotational inertia of the sphere.

In the pendulum on a cart problem, the pivot point of the pendulum is always known to be at the hinge on the cart. In Vertigo's case, this point is not so easily determined. Its center of rotation is a function of the body mass, sphere mass, inertia, rotational inertia, velocity and angular rates. When control is being applied, this point is shifting wildly and is extremely difficult to determine. To help illustrate how complex the pivot drift is, Figure A.6 shows Vertigo's response to the most simplistic scenario: an uncontrolled fall. The figure shows snapshots of Vertigo's central axis at ten degree intervals as it falls uncontrolled. Here it is assumed that the robot does not ever fall off the sphere and that only the sphere is constrained by the ground. The pivot path can be traced out by the inner arch that is formed by the tangential lines. The reason for this path is that Vertigo's center line must pass through the center of the sphere, while at the same time, the center of the sphere is moving as Vertigo falls over and it rolls along the ground. The translation and rotation generate a path that is not characteristic of any standard polynomial or trigonometric function. This thought experiment is for the most straightforward motion possible,

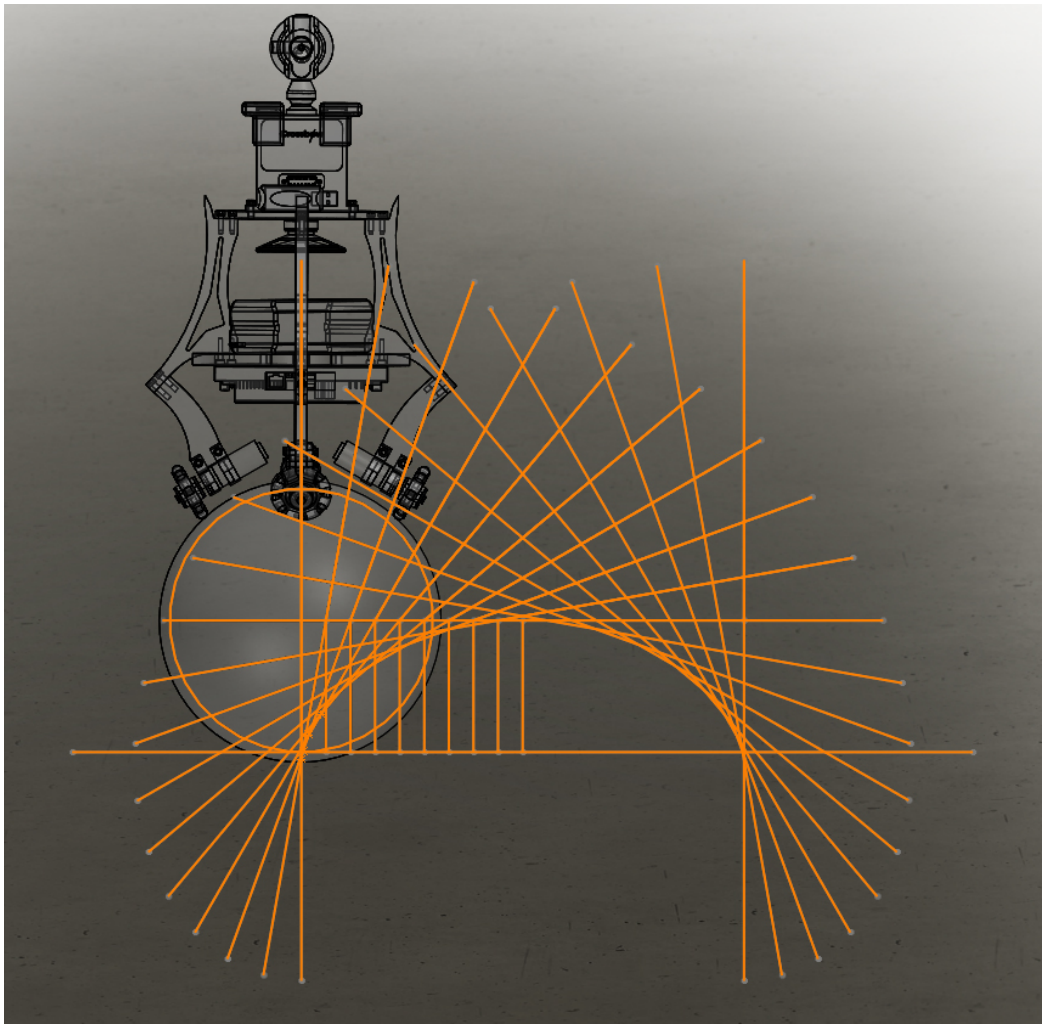


Figure A.6: Uncontrolled pivot drift.

as soon as control is applied the sphere and body of the robot do not rotate as one, and the complexity is magnified.

A.7 Sphere Properties

The properties of the sphere have been established as a critical component of this system, this is the second dominant departure from the classic pendulum on a cart problem. Exploring exactly how it influences the dynamics is consequently important. Beyond features like material and texture, which were discussed previously in the Design chapter, this section looks at how the physical properties of the sphere, such as mass and diameter, affect the system.

If, for a moment, the mass and inertia of the sphere are somehow assumed constant, and only the diameter is considered, it is found that balancing on a larger sphere is easier than balancing on a smaller one. This phenomenon is often falsely attributed to a gear reduction in the control, where the actuator is the first gear, the sphere is a middle gear, and the ground is the final gear, or pinion. This is not true however. As with any planar multi gear linkage, the only gears that contribute to the overall ratio are the first and last, or in this case the wheel and ground. The sphere size has no affect on this because as the wheel rotates the sphere, that arc length is conveyed exactly to the next gear (ground), resulting in identical translation across the ground for any size sphere. The only way to change Vertigo's gearing past the motor's gear head is to change the diameter of the wheels. That being said, a larger sphere is still easier to balance on. The reason for this is apparent when considering the uncontrolled dynamics. As Vertigo tips over, it is also rotating the sphere. A sphere with a larger diameter will travel along the ground farther per degree of rotation. Since this translation is in the same direction as the control needed to balance, a larger diameter sphere is naturally easier to stabilize. In addition, a larger sphere will raise the center of gravity which is beneficial as will be discussed later in this appendix.

Inertia is a property of matter which states that more massive objects have greater resistance to change in motion. Rotational inertia (moment of inertia) is a similar principle that deals with resistance to change in rotation rate, and is a function of mass and geometry. To investigate the affects of sphere mass, the size can be assumed fixed, so both forms of inertia will increase with mass. If we consider an extremely massive sphere, it will be extremely resistant to change in motion, which means Vertigo will be able to balance much more easily. It would be analogous to a fixed sphere, where Vertigo would simply have to climb to the top, and then no more control would be necessary. The fatal drawback is that greater inertia requires more control input to accelerate. That is, when navigation is desired, the system has to work very hard to move the sphere. Now, a near massless sphere would require very responsive control to balance, because there would be little inertial resistance to change. However, navigation would use comparatively less control. Besides pivot drift, a massless sphere would cause Vertigo to behave very similarly to the pendulum on a cart because sphere inertias would have no effect.

It is worth noting here that Vertigo is designed to be a controls experiment, so it is important to select a sphere with properties that balance the challenges of control with practical feasibility.

A.8 Weight Distribution

Since inverted pendulums, by definition, have centers of gravity above their pivot point, and are known to be unstable, it is a common notion that lowering the center of gravity will make it easier to control. To investigate this idea, assume the cart in Figure A.5 is fixed, making it a simple uncontrolled inverted pendulum. The

equation of motion for the angle θ would then be $\ddot{\theta} = \frac{g}{l} sin(\theta)$, where the length of the pendulum is inversely proportional to the acceleration. Thus, an inverted pendulum with a higher center of gravity will fall more slowly, giving the controller more time to respond and making it easier to control.

These simple equations provide cogent evidence, but there is a more intuitive way of understanding this concept. To control the inverted pendulum, it must be rotated by a torque, $T = Fl$. If the control input is a force F at the base, then the length l is the lever arm for the torque. When the force is applied, both linear and rotational motion will ensue at the center of gravity. A longer lever arm means less control force is required to impart the same torque. Additionally, smaller F will result in smaller linear motion, so the pendulum will sway less when balancing. This is easily seen when balancing a baseball bat, or broom: it is much less challenging when the heavier end is at the top. Similarly, the rotational inertia has an effect on the system. If the mass is more concentrated, the rotational inertia is reduced, and less input is required to rotate the pendulum.

Appendix B

Powering Vertigo and Qwerk

Through the evolution of Vertigo, and as its demands changed, several powering methods were used. Currently, only two remain in use.

- LiPo battery packs
- Tethered power supply

LiPo Battery Pack:

Zippy-R 4150 mAh, 11.1 V, 3s1p (X2)

This method is best for mobile powering. There are two identical battery packs, allowing a charged pack to be on hand at all times. Both packs bundle two 4150 mAh, 11.1 V bricks that are charged separately (see charging instructions below). They are connected in parallel so they deliver their original voltage for twice as long. The batteries were fitted with Deans connectors that enable quick, secure and high performance connection to the Y harness that wires the two bricks in parallel. To protect Qwerk, and insure that the minimum limits of the battery packs are not breached, a 9.0V LiPo battery voltage cut-off circuit was fitted between the power supply,

and the power input for Qwerk. (http://www.rctoys.com/rc-toys-and-parts/DF-DIV-BATTCUT/RC-PARTS-WIRELESS-VIDEO-BATTERY-ACCESS.html)

Battery Charging: For single 4150 mAh, 11.1 V, LiPo battery pack.

1. Plug single (must charge both of the packs individually) battery into TRITON 2 charger through the Equinox interface.
2. On the charger, click Battery Type to select LiPo charge, and then choose 4150 mAh, 11.1 V.
3. Click the Equinox interface box to switch to interface mode.
4. Press dial on charger and hold to initiate the charging sequence.

Charging takes roughly an hour and a half per pack. Always charge both bricks before use. It is a good idea to charge the pack immediately after they run out to ensure that it will be ready when needed.

NOTE: The charger has a safety timer that may expire before full charge is reached. It will say why it stopped on the display. If it timed out, simply proceed as though you were charging it again, and it will finish.

Tethered Power Supply:

Input: AC 100-230 V, 0.7-0.4 A, 50/60 HZ

Output: DC 12V, 1.6 A

This method is best for stationary powering (programming, diagnostics, etc.). Programming of any type should really be done on this power supply to minimize the risk of undesired loss of power which may potentially damage Qwerk's hardware

or software. This supply was lifted off of an old computer. The wiring was cleaned up, removing all unnecessary leads, and it was fitted with a 12 foot power tether that connects directly to Qwerk's power input. Also added was a master power switch and green LED that indicates the unit is on.

Appendix C

Connection to Vertigo

Method 1: Through TERK program launched from site

1. Go to www.terk.ri.cmu.edu/software/index.php
    (a) Choose program and launch it
2. Close internet
3. Plug router in
4. Power Vertigo (Qwerk)
    (a) Qwerk will automatically connect
        i. LED 6 will blink when connection is established
        ii. It is normal for LEDs 0, 1, 2 and/or 3 to be blinking or solid
            A. I think LED 3 blinking means it is connected to another comp.
5. Connect computer to LAIRS router

   (a) Wirelessly or cable in

      i. Password: “lairs”

   (b) Can check the router status by typing 192.168.1.1 into web browser (username: leave this blank, password: ”admin”)

      i. StatusLocal NetworksDHCP Clients Table

      ii. Should see ip addresses of both the computer and Qwerk

6. In the launched software, click the [Connect] button

   (a) Connect directly to a peer

   (b) Type in ip address of Vertigo and connect

Vertigo specific instructions are presented first, then the Terk given instructions for installation of programs and running code.

1. Open JCreator LE version 4.50.010

2. File/open workspace

   (a) VERTIGO/QWERK/MyFirstVertigo/terk-client-MyFirstRobot/

   (b) Choose workspace

3. Choose the program you would like to run

4. Compile the project by either hitting F7 or going to Build->’Compile Project’ in the top menu

5. Run the project by going to Build->’Execute File’ in the top menu. Note that you must execute the file, NOT the project.

6. Running the program will open a graphical interface. To connect to a robot, click on the [Connect] button.

7. From here, follow steps 5 and 6 from the previous section (Through TERK Program Launched From Site).

THESE ARE THE PROVIDED INSTRUCTIONS FOR INSTALLING AND CONNECTING THROUGH JAVA

ROBOT CLIENT QUICKSTART

———————-

Purpose

——-

This document describes how to install, compile, and run the MyFirstRobot program.

It also briefly describes important other files in this folder.

Installation

————

* Create a directory for the MyFirstRobot program and its associated files.

* Unzip the terk-client-MyFirstRobot.zip file into the directory you just created.

* You will need to have Sun's Java SE JDK 5.0 installed. If it's not already installed, you can download it at:

http://java.sun.com/javase/downloads/index.jsp
Follow their instructions for installing it on your machine.

Compiling and Running the MyFirstRobot Program in JCreator
(RECOMMENDED METHOD)

JCreator is a freeware Java IDE. A project file, MyFirstRobot.jcp, is available in the MyFirstRobot folder with all of the correct configuration settings to quickly begin compiling and running the MyFirstRobot program.

* Install JCreator LE from http://www.jcreator.com/download.htm

* Open the file MyFirstRobot.jcp. The JCreator IDE should start.

* In the File View of the IDE, open MyFirstRobot.java for editing by double clicking on it.

* Compile the project by either hitting F7 or going to Build->'Compile Project' in the top menu.

* Run the project by going to Build->'Execute File' in the top menu. Note that you must execute the file, NOT the project.

* Running the program will open a graphical interface. To connect to a robot, click on the "Connect" button. A connection dialog will pop up and you will have to select whether you are connecting to the robot via the CMU-based relay (relay mode)

or if you will connect over a local network by entering the robot's IP address (direct connect). For more information about these two modes of operating your qwerk, visit http://www.terk.ri.cmu.edu/projects/qwerk-overview.php. Make your selection and click 'Next':

- If you entered direct connect mode, you will now be prompted to enter the IP address of the robot. Do so and click 'Connect', then click 'Finish'. You should now be connected to the robot.

- If you entered relay mode, you will be prompted to enter your TeRK login and password. Enter your login, click the 'Login' button, and then click 'Next' to go to the next page in the connection wizard. You will see a list of available robots. Highlight the appropriate robot by clicking on it once. Then click the 'Connect' and the 'Finish' buttons. You should now be connected to the robot.

* Once you are connected to a robot, you can begin displaying an image stream from the robot's onboard camera by clicking on "Start Video". Once started, you can pause video at any time by hitting "Pause Video", and capture images from the video stream by clicking "Save Picture". To run your program, hit the 'Play' button. To stop your program, hit 'Stop'.

* Several other example files are available and visible in the file view. They are Headturner.java, Photovore.java, hearingTest.java, and soundTest.java. You can compile and run these as you did for MyFirstRobot.java.

Compiling and Running the MyFirstRobot Program from Command Line

---

* Open a command prompt and set the current directory to the directory you created during installation.

* Type "compile" (no quotes) at the command prompt to compile the program. It will take a few seconds to compile, and will print out any syntax errors that may occur.

* If there are no compilation errors, type "run" (no quotes) to run the program.

* Running the program will open a graphical interface. To connect to a robot, click on the "Connect" button. A connection dialog will pop up and you will have to select whether you are connecting to the robot via the CMU-based relay (relay mode) or if you will connect over a local network by entering the robot's IP address (direct connect). For more information about these two modes of operating your qwerk, visit http://www.terk.ri.cmu.edu/projects/qwerk-overview.php. Make your selection and click 'Next':

- If you entered direct connect mode, you will now be prompted to enter the IP address of the robot.

Do so and click 'Connect', then click 'Finish'. You should now be connected to the robot.

- If you entered relay mode, you will be prompted to enter your TeRK login and password.

Enter your login, click the 'Login' button, and then click 'Next' to go to the next page in the

connection wizard. You will see a list of available robots. Highlight the appropriate robot

by clicking on it once. Then click the 'Connect' and the 'Finish' buttons. You should now

be connected to the robot.

* Once you are connected to a robot, you can begin displaying an image stream from the robot's onboard camera by clicking on "Start Video". Once started, you can pause video at any time by hitting "Pause Video", and capture images from the video stream by clicking "Save Picture". To run your program, hit the 'Play' button. To stop your program, hit 'Stop'.

Important Locations in this Folder

---

javadocs Folder: Documentation for every method in the classes included in this project.

API Reference.pdf: This file contains descriptions and sample usages of all methods used for controlling the Qwerk, accessing robot sensor data, and using the GUI. Read this document or the javadocs before writing any programs!!

MyFirstRobot.java: Skeleton starter file.

MyFirstCreate.java: Skeleton started file for controlling the iRobot Create.

Examples: Example programs which demonstrate some of the available robot control methods.

All examples are commented and contain a header with a description of the program's operation.

MyFirstRobot.jcp: Jcreator project file. Double click to open JCreator and the MyFirstRobot project.

MyFirstRobot.jcw and .jcu: Other Jcreator files.

compile.bat: Batch file for compiling MyFirstRobot.java from the command line

run.bat: Batch file for running MyFirstRobot.java from the command line

clean.bat: Batch file for removing any files generated by using compile or run.bat

RobotClient: Folder containing classes for accessing and controlling the robot and GUI

- RobotClient.java: Contains all methods for controlling and accessing the robot, as well as the GUI

- RobotClientGUI.java: Specifies the properties of the GUI.

- SimpleRobotClient.java: Wrapper class for RobotClient.java - can be used interchangeably with RobotClient.java

- CreateClient.java: Class containing methods for controlling the iRobot Create or iRobot Roomba. See the Create recipe at www.terk.ri.cmu.edu for more information.

RSSReaders: Folder containing classes for reading RSS feeds from the internet:

- RSSReader.java: Generic class for reading feeds

- WeatherReader.java: Class for reading weather data for any US city, courtesy of www.wunderground.com feeds

TTS: Folder containing class for generatic speech from text

- TTS.java: Generic class for generating text to speech

==========

Method 3: Through MATLAB

Instantiating Objects from Java Classes in Matlab

NOTE: Steps 2 and on may not be needed if you are using a dynamic Java path (see **).

Versions used: Windows Vista Business

Java - jdk1.6.0_06

Matlab 7.4.0 (R2007a)

1. Matlab comes with a version of Java that it uses instead of the one installed on your computer. You can check the version with the **version –java** command in command window. You should update Matlab's java version to the one you have been compiling your classes with.

   (a) This can be done by right clicking on '*My Computer*' and going into '*Advanced system settings*'.

   (b) Next, click on the '*Advanced'* tab then click on the '*Environmental Variables'* button

   (c) Under the '*System Variables*' *section,* add a new variable MATLAB_JAVA with the following value '*C:\Program Files\Java\jdk1.6.0_06\jre*' (this was it in my case) or a value that corresponds to the version of Java that you are running.

   (d) Enter **version –java** to ensure that the version is updated (restarting Matlab may be required)

2. Enter **edit classpath.txt** into the command window and add the directory where your class files are located in a new line.

C:\Users\Jon\Desktop\VERTIGO\QWERK\MyFirstVertigo

\terk-client-MyFirstRobot\

1. Try constructing a java object:

Example of constructing a java object in Matlab using two methods:

Method 1:

> > j=javaObject('AddNum',3.2,4.2)

> > j.writeValue()

7.4 Method 2:

> > q=AddNum(3.4,1.3)

> > q.writeValue()

4.7 This was first tested this with a .class file generated from the javac compiler from the command prompt. Files generated from an IDE such as JCreator or Eclipse were then successfully tested as well. This method has not yet been tested with stuffing the class into a package.

Sample class:

```
public class AddNum {
    private double myValue;
    public AddNum(double a,double b){
    myValue=a+b;
    }
    public void writeValue(){
    System.out.println(myValue);
    }
}
```

Instructions **After** Initial Installation and Setup

1. Open MATLAB

2. Open vertigo_qwerk folder (in MATLAB)

3. Enter Ts = 0.1 in the command window
4. Run vertigo_nonlinear.m
5. Open MyFirstRobot.m, make sure the directory locations are all correct (see code below *)
   (a) The classpath.txt file altered above in the initial setup, must be returned to its original state (see reference **, a section copied from the rn.pdf in the QWERK dir.).
6. Run MyFirstRobot.m, GUI should pop up just like other connection methods
   (a) Click [Connect]
   (b) Choose to connect directly to a peer
   (c) Enter Vertigo's IP address (something like 192.168.1.100)
   (d) Click [Finish], then [Play], and minimize the GUI
7. Now all of the Terk commands can be used in MATLAB just as they were used in JCreator for Java.
8. Run vertigo_java_vicon_loop.mdl, found in the vertigo_simulink directory

*

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% setting the path to include required %dir javaaddpath('C:\Users\Jon\Desktop \VERTIGO\QWERK\MyFirstVertigo\terk-client-MyFirstRobot');

% setting the path to terk.jar also. terk.jar is where the motor,

% connection and other commands are located javaaddpath('C:\Users\Jon\Desktop\VERTIGO\QWERK\MyFirstVertigo\terk-client-MyFirstRobot\terk.jar');

%add text to speech functionality

%javaaddpath('C:\Users\Jon\Desktop\VERTIGO\QWERK\MyFirstVertigo\terk-client-MyFirstRobot\TTS')

%javaaddpath('C:\Users\Jon\Desktop\VERTIGO\QWERK\MyFirstVertigo\terk-client-MyFirstRobot\RobotClient');

%applicationTitle=java.lang.String('Woof');

%ipaddress=java.lang.String('192.168.1.100');

%myRobot = RobotClient.RobotClient('applicationTitle');%,ipaddress);

%myRobot = RobotClient.SimpleRobotClient('trial');

%%%%%%%%%%%%%%%%%%%%%%%%%%

**

Java Interface Adds Dynamic Java Class Path (excerpt from rn.pdf in the QWERK dir.)

MATLAB loads Java class definitions from files that are on the Java class path. The Java class path now consists of two segments: the *static path*, and a new segment called the *dynamic path*. The static path is loaded from the file classpath.txt at the start of each MATLAB session and cannot be changed without restarting MATLAB. This was the only path available in previous versions of MATLAB. Thus, there was no way to change the Java path without restarting MATLAB. The dynamic Java class path can be loaded at any time during a MATLAB session using the javaclasspath function. You can define the dynamic path (using javaclasspath), modify the path

(using javaaddpath and javarmpath), and refresh the Java class definitions for all classes on the dynamic path (using clear java) without restarting MATLAB. See the function reference pages for more information on how to use these functions.

The javaclasspath function, when used with no arguments, displays both the static and dynamic segments of the Java class path:

javaclasspath

STATIC JAVA PATH

D:\Sys0\Java\util.jar

D:\Sys0\Java\widgets.jar

D:\Sys0\Java\beans.jar

DYNAMIC JAVA PATH

User4:\Work\Java\ClassFiles

User4:\Work\Java\mywidgets.jar

You can read more about this feature in the sections, "The Java Class Path" and "Making Java Classes Available to MATLAB" in the External Interfaces documentation.

Appendix D

Changing Qwerk's Embedded Code

Special thanks to Mark Tjersland for all his help in finally cracking this.

This document goes through how to change Qwerk's embedded code. More specifically, the hardcoded gains for the PIDV controller, and the trajectory generation function that the PIDV controller tracks. This was necessary to attain the responsiveness demanded by Vertigo. It details how to acquire the code, manipulate it, compile it, delete the existing code, replace it, and reinitialize Qwerk. It also lists all the required programs for these steps and how they can be acquired.

This procedure is loosely based on the outline given in the **Qwerk Development Guide (QDG)** which can be found at:
http://terk.svn.sourceforge.net/viewvc/terk/embed/trunk/share/docs/QwerkDevelopmentGuide.html#BuildingtheQwerkSource
The relevant excerpts can also be found at the end of these instructions.

Programs:

For this procedure you will need root access to in Linux so the SENS accounts

through the school will not work. You may need to install a virtual machine on your computer. This can be done by installing Sun xVM VirtualBox (free online). Also, ubuntu-6.06.1-desktop-i386 is needed. More current versions will not compile the code correctly. You will also need to install the drivers for the USB-Serial adapter. Remember the port that the adapter is plugged into when you do this because it will always need to be plugged into the same port when connecting to and altering the embedded Qwerk code.

Procedure:

Compiling Firmware:

Open **Sun xVM VirtualBox** (NOTE: Remove all flash drives, SD cards and external hard drives. Also, the blue USD to serial adapter must be plugged in to the port it was in during installation of drivers. See figure below for current location on the editing computer.)

Click [**state**]. It will boot up then ask for username and password.

Username: **jwmissel** (these specifics will change with the user)

Password: **password**

In Linux, go to **applications accessories terminal**.

In terminal type in: **cd terk/embed/trunk/share/src/terkapi**

Then type: **gedit Client.cpp &** This is case sensitive. Executing this command will open up a new window with Qwerk's source code in it. You will now need to find the lines with the gains for the motor controller and change them as desired. To make this faster, click [**find**] and search for the keyword "**gain**." After changing them click [**save**].

NOTE: The original gains were (100, 0, 500, 0). These, with the original tra-

jectory, required about 10 seconds for the motors to accelerate. Much too slow of a response; to mitigate this, the P and D gains should be increased and the I gain should remain low. The combination (500, 0, 1000, 0) was too high and caused spastic motor behavior. This was lowered until they worked properly again at (150, 0, 600 ,0). Here the response took about 6 seconds. Still too slow.

For changing the acceleration type:
cd terk/embed/trunk/share/src/libqwerk then **gedit qemotortraj.cxx**. Then multiply the acceleration by a constant to dilate it's affect. It is currently multiplied by 200 to achieve a slope that is near to a step function. [**save**] The following section picks up in the QDG under the section: Building the Qwerk Source.

In terminal, run the commands (one at a time) under Building the Qwerk Source:

cd ~/terk/embed/trunk/share

make

cd ~/terk/embed/trunk/cirrus-arm-linux-1.0.4

make edb9302 (NOTE: may take about 8 mins to run**)**

cd ~/terk/embed/trunk/cirrus-arm-linux-1.0.4/edb9302

for i in ramdisk.gz zImage optfs.out; do
(sudo cp –backup=t $i /var/www); done

Then just type in the password: **password** DOESN'T ALWAYS ASK FOR THIS
At this point the new firmware has been compiled, but is not yet installed on

Qwerk.

Installing Firmware:

All programs and windows from above should still be open.

Cable-in to router: Both Vertigo (Qwerk) and your computer MUST be cabled in!

Connect to serial port: You must connect your computer to Qwerk through the serial port (UART1 on Qwerk). Plug the USB end of the blue USB Serial Adapter into your computer. Connect the serial end to the gray serial to RJ11 (phone jack) adapter, then connect that to the UART1 jack on Qwerk. See figure below for visual reference, the red line shows the lines of connection under consideration.

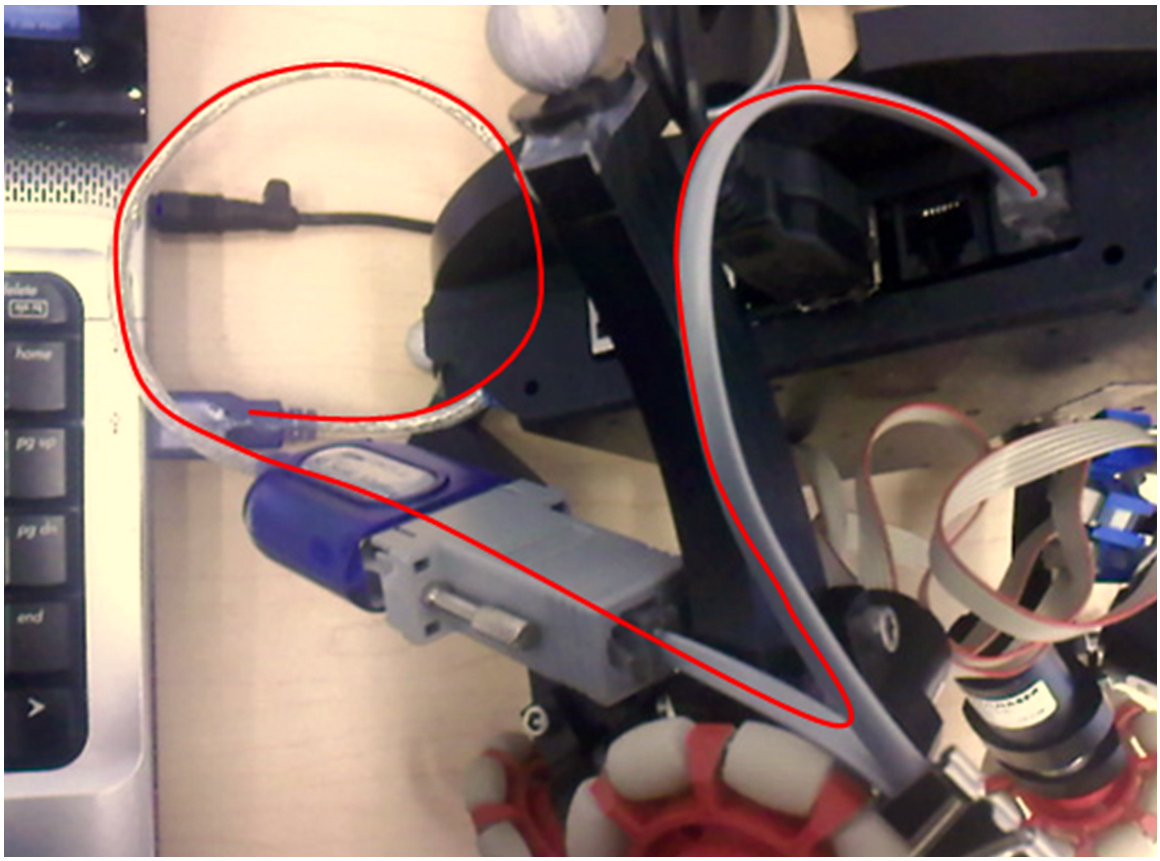


Figure D.1: Wiring for access to Qwerk.

Type **sudo minicom** in the Terminal window. This should bring up a Welcome to minicom 2.1 prompt.

Turn off Qwerk.

Click on Terminal (to make it the active window).

Turn on Qwerk AND hit **[ctrl]+[C]** when you see text appear in Terminal. There is a one second window you have to do this in. Hitting it a few times to make sure you got it won't hurt anything.

Press **[enter]** if instructed.

If this part has been executed properly, it would end up giving you the RedBoot> command prompt in terminal. This is where you will enter the following commands. In terminal, run the commands in the QDG under Loading the Linux Images:

load -r -v -b 0x1000000 -m http /optfs.out

load -r -v -b 0x800000 -m http /ramdisk.gz

load -r -v -b 0x80000 -m http /zImage

fis create -b 0x800000 -l 0x300000 ramdisk

Answer **y** to the posed question for continuation.

fis create -b 0x80000 -l 0x180000 -e 0x80000 zImage

Answer **y** to the posed question for continuation.

Then run the command for version 1.1 Qwerk (the first option in the QDG document):

fis create -b 0x1000000 -l 0x300000 optfs

Answer **y** to the posed question for continuation.

Type **reset** in Terminal. NOTE: Don't hit [ctrl]+[C] this time because you want it to reboot.

Wait and press [**Enter**] when asked.

At this point, the firmware has been replaced and Qwerk is just like it was out of the package, except of course for the embedded code that have been changed.

Now you have to change its connection settings back to where they were for communication.

Change Qwerk's connection settings:

With everything still plugged in and connected as it was before, enter Qwerk's **IP address into the web browser**. Under **General Configuration** select **Direct-Connect**. Then click [**Save**]. Then under **Wireless Configuration**, check **Set the SSID Manually** and make the settings as follows:

Network name (SSID): **LAIRS**

Data encryption: **WEP**

Network key: **B4A8BFD5F4**

Max tune to retry this SSID: **60**

Max unsuccessful attempts: **2**

Click [**Save and Restart**].

Reboot Qwerk. After reboot, led 6 should be blinking, indicating that you are connecting directly. If led 7 is blinking, it means that it is trying to connect through a relay server. This is what was just changed in the previous step.

Everything should be back to normal and running properly with the new gains and trajectory.

Qwerk Development Guide (QDG): Only relevant sections are included here.

This document describes all the steps required to develop software for the Qwerk.

Blocks with a grey background represent what you'll see in the command prompt. **Bold** text within those blocks is text you enter.

Building the Qwerk Source

Compile the share tree first. The share tree contains all the TeRK-specific code. You can safely ignore all the "might be used uninitialized in this function" warnings reported during the build.

$ **cd ~/terk/embed/trunk/share**

$ **make**

make -C ./src

make[1]: Entering directory '/home/YOUR_USERNAME/terk/embed/trunk /share/src'

make -C IceE-1.0.0 all

make[2]: Entering directory '/home/YOUR_USERNAME/terk/embed/trunk /share/src/IceE-1.0.0'

making all in src

make[3]: Entering directory '/home/YOUR_USERNAME/terk/embed/trunk /share/src/IceE-1.0.0/src'

making all in IceE

make[4]: Entering directory '/home/YOUR_USERNAME/terk/embed/trunk /share/src/IceE-1.0.0/src/IceE'

rm -f ../../include/IceE/BuiltinSequences.h BuiltinSequences.cpp

slice2cppe –ice –include-dir IceE –dll-export ICE_API -I../../slice ../../slice/IceE /BuiltinSequences.ice

mv BuiltinSequences.h ../../include/IceE

rm -f ../../include/IceE/FacetMap.h FacetMap.cpp

slice2cppe –ice –include-dir IceE –dll-export ICE_API -I../../slice ../../slice/IceE /FacetMap.ice

...

~/terk/embed/trunk/share

Writable filesystem successfully created!

$

Building the share tree may take a while, but should eventually finish with the report "Writable filesystem successfully created!". The writeable filesystem it created is a file named optfs.out and is located in the ~/terk/embed/trunk/cirrus-arm-linux-1.0.4/edb9302 directory.

The build also copies over the opt directory from which the optsfs.out image was created (this is necessary since, as we'll see next, the build for the cirrus-arm-linux tree also creates the optfs.out image by repacking the opt directory that it finds within its ~/terk/embed/trunk/cirrus-arm-linux-1.0.4/edb9302 directory).

Now build the cirrus-arm-linux tree. This builds three filesystem images for the Qwerk:

$ **cd ~/terk/embed/trunk/cirrus-arm-linux-1.0.4**

$ **make edb9302**

make[1]: Entering directory '/home/YOUR_USERNAME/terk/embed/trunk/cirrus-arm-linux-1.0.4/edb9302'

Creating root filesystem...

Creating linux source tree...

Configuring linux...

Building linux modules...

Configuring module-init-tools...

Building module-init-tools...

...

Building linux zImage...

Copying linux...

Creating redboot source tree...

Building redboot...

Installing redboot...

Building IceE...

Building firmware...

Building r/w filesystem...

make[1]: Leaving directory '/home/YOUR_USERNAME/terk/embed/trunk/cirrus-arm-linux-1.0.4/edb9302'

$

This will generate three important files: zImage, ramdisk.gz, and optfs.out. All of these are in the edb9302 directory under the cirrus-arm-linux tree.

Building the cirrus-arm-linux tree will likely take a while, but luckily it's not something you'll have to build very often. The code in the cirrus-arm-linux tree rarely changes, so you'll most likely build it once to create the zImage and ramdisk.gz images and then you'll just do builds of the share tree (which does change often) which creates the optfs.out image.

Note that although the cirrus-arm-linux build generates a (new) optfs.out image, all it's really doing is repacking the opt directory which was copied into ~/terk/embed /trunk/cirrus-arm-linux-1.0.4/edb9302 by the share tree build above. It repacks the image by calling the buildoptfs.sh script. In any case, it should be the same as the one created by the share tree build.

We're now almost ready to download the image files to the Qwerk! We'll first need to put the images in the Apache's web root directory so your machine can serve them to the Qwerk. To do so, copy the three image files to the /var/www directory. You can either do so in the usual way, or by doing the following which creates backups of any images which currently exist (this doesn't matter for this initial copy, of course, but you may find it nice to use in the future):

$ **cd ~/terk/embed /trunk/cirrus-arm-linux-1.0.4/edb9302**

$ **for i in ramdisk.gz zImage optfs.out; do**

(sudo cp –backup=t $i /var/www); done

$

Network Setup

Configure your network as described in the Network Setup section of the Upgrading Firmware document on the TeRK web site. Since your computer will be serving the firmware images to the Qwerk, you must have your network configured as shown in figures 4 or 5 of the Network Setup document.

Installing Firmware Images

To install Linux on the Qwerk, we'll connect to it via the serial cable using Minicom, load in three images, and then create a boot script.

Installing and Configuring Minicom

We first need to install minicom:

$ **sudo apt-get install minicom**

Reading package lists... Done

Building dependency tree... Done

Recommended packages:

lrzsz

The following NEW packages will be installed:

minicom

0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.

Need to get 155kB of archives.

After unpacking 913kB of additional disk space will be used.

Get:1 http://us.archive.ubuntu.com dapper/main minicom 2.1-10 [155kB]

Fetched 155kB in 4s (34.0kB/s)

Selecting previously deselected package minicom.

(Reading database ... 86463 files and directories currently installed.)

Unpacking minicom (from .../minicom_2.1-10_i386.deb) ...

Setting up minicom (2.1-10) ...

Now run minicom

$ **sudo minicom**

Welcome to minicom 2.1

OPTIONS: History Buffer, F-key Macros, Search History Buffer, I18n

Compiled on Nov 5 2005, 15:45:44.

Press CTRL-A Z for help on special keys

We'll now configure minicom to be able to talk to the Qwerk. Minicom should be set to use 57600 baud, 8-N-1 (8 data bits, no parity, 1 stop bit), and no software or

hardware flow control. Type CTRL-A Z to bring up the main menu:

```
+--------------------------------------------------------+
|Minicom Command Summary                                 |
|                                                        |
|Commands can be called by CTRL-A <key>                  |
|                                                        |
|Main Functions                 Other Functions          |
|                                                        |
|Dialing directory..D run script (Go)....G |Clear Screen.....C |
|Send files.........S Receive files......R |cOnfigure Minicom..O |
|comm Parameters....P Add linefeed.......A |Suspend minicom....J |
|Capture on/off.....L Hangup.............H |eXit and reset.....X |
|send break.........F initialize Modem...M |Quit with no reset.Q |
|Terminal settings..T run Kermit.........K |Cursor key mode....I |
|lineWrap on/off....W local Echo on/off..E |Help screen........Z |
||scroll Back........B                                   |
|                                                        |
|Select function or press Enter for none.                |
|                                                        |
|Written by Miquel van Smoorenburg 1991-1995             |
|Some additions by Jukka Lahtinen 1997-2000              |
|i18n by Arnaldo Carvalho de Melo 1998                   |
+--------------------------------------------------------+
```

Enter O to configure minicom:

```
+------[configuration]------+
|Filenames and paths        |
```

|File transfer protocols |

|Serial port setup |

|Modem and dialing |

|Screen and keyboard |

|Save setup as dfl |

|Save setup as.. |

|Exit |

————————————-|

Now use the arrow keys to highlight "Serial port setup" and type ENTER to choose it. You should now see the following menu:

————————————————————-

|A - Serial Device : /dev/tty8 |

|B - Lockfile Location : /var/lock |

|C - Callin Program : |

|D - Callout Program : |

|E - Bps/Par/Bits : 38400 8N1 |

|F - Hardware Flow Control : Yes |

|G - Software Flow Control : No |

||

|Change which setting? |

————————————————————-

First type A to set the serial device. The device name to use depends on your computer. For example, one of my machines only has one serial port, so I set the device to /dev/ttyS0. Another (a MacBook running Ubuntu under VMWare Fusion) doesn't have any serial ports, so I use a GWC UC320 USB-to-serial adapter which Ubuntu recognizes as /dev/ttyUSB0. The name of your serial port device may be

different. When you're done editing the device, type ENTER to save it.

```
————————————————————-
|A - Serial Device : /dev/ttyS0 |
|B - Lockfile Location : /var/lock |
|C - Callin Program : |
|D - Callout Program : |
|E - Bps/Par/Bits : 38400 8N1 |
|F - Hardware Flow Control : Yes |
|G - Software Flow Control : No |
||
|Change which setting? |
————————————————————-
```

Now type E to set the Bps/Par/Bits. Doing so will open a new menu that should look like this:

```
————————[Comm Parameters]————————-
||
|Current: 38400 8N1 |
||
|Speed Parity Data |
||
|A: 300 L: None S: 5 |
|B: 1200 M: Even T: 6 |
|C: 2400 N: Odd U: 7 |
|D: 4800 O: Mark V: 8 |
|E: 9600 P: Space |
|F: 19200 Stopbits |
```

|G: 38400 W: 1 |

|H: 57600 X: 2 |

|I: 115200 Q: 8-N-1 |

|J: 230400 R: 7-E-1 |

||

||

|Choice, or <Enter> to exit? |

Enter H to select 57600 baud, then Q to select 8-N-1. As you do so, you should see the line at the top which displays the current settings change. When you're done, type ENTER to close the Comm Parameters menu. You should now see that the Bps/Par/Bits value has changed:

|A - Serial Device : /dev/ttyS0 |

|B - Lockfile Location : /var/lock |

|C - Callin Program : |

|D - Callout Program : |

|E - Bps/Par/Bits : 57600 8N1 |

|F - Hardware Flow Control : Yes |

|G - Software Flow Control : No |

||

|Change which setting? |

Finally, type F to turn off Hardware Flow Control. The settings should now look like this:

|A - Serial Device : /dev/ttyS0 |

|B - Lockfile Location : /var/lock |

|C - Callin Program : |

|D - Callout Program : |

|E - Bps/Par/Bits : 57600 8N1 |

|F - Hardware Flow Control : No |

|G - Software Flow Control : No |

||

|Change which setting? |

———————————————-

Type ENTER to exit this menu.

Now that we're back at the configuration menu, use the arrow keys to highlight the "Save setup as dfl" option and type ENTER to select it.

Now use the arrow keys to highlight the "Exit" option and type ENTER to select it.

Finally, quit minicom by typing CRTL-A x.

Minicom is now configured to talk to Qwerks!

Connecting to the Qwerk Using Minicom

Run minicom:

$ **sudo minicom**

Welcome to minicom 2.1

OPTIONS: History Buffer, F-key Macros, Search History Buffer, I18n

Compiled on Nov 5 2005, 15:45:44.

Press CTRL-A Z for help on special keys

We're now ready to turn on the Qwerk. Once we do so, it'll spit out some system info and then pause for 1 second before executing its boot script. We don't want it to execute the boot script yet, so we're going to type CTRL-C to abort it.

Go ahead and turn on the Qwerk, but type CTRL-C immediately afterwards.

It should look something like this:

+Ethernet eth0: MAC address 00:dc:6c:7d:6b:25

IP: 192.168.1.120/255.255.255.0, Gateway: 192.168.1.1

Default server: 192.168.1.103, DNS server IP: 192.168.1.1

RedBoot(tm) bootstrap and debug environment [ROMRAM]

Non-certified release, version v2_0 - built 13:09:18, Mar 21 2006

Platform: Cirrus Logic EDB9302 Board (ARM920T) Rev A

Copyright (C) 2000, 2001, 2002, Red Hat, Inc.

RAM: 0x00000000-0x02000000, 0x00041fa8-0x01fdd000 available

FLASH: 0x60000000 - 0x60800000, 64 blocks of 0x00020000 bytes each.

== Executing boot script in 1.000 seconds - enter ^C to abort

^C

RedBoot>

You should now see a "RedBoot>" prompt.

Pay attention to the IP, Gateway, and Default Server IP addresses. The IP address is the address of the Qwerk, the Gateway is your router, and the Default Server is the HTTP server (i.e. your machine). If the IP addresses are correct, then you can skip down to the Loading the Linux Images section. Otherwise, follow the directions in the Configure the Qwerk For Your Network section to set them for your network.

Configure the Qwerk For Your Network

If the IP, Gateway, and Default Server IP addresses (displayed in minicom when you first power on the Qwerk) are incorrect, then you won't be able to download the images from your HTTP server. The IP address is the address of the Qwerk, the Gateway is your router, and the Default Server is the HTTP server (i.e. your ma-

chine). I usually go ahead and set the DNS Server IP Address, too, which should in most cases just be the same as the Gateway IP Address.

Execute the fconfig command to set the IP, Gateway, and Default Server IP addresses. Start by entering true for whether the script should run at boot, and then enter the script exactly as shown below. Once you've entered the boot script, enter an empty line to terminate the script editing mode. Continue on through the other values and enter the correct IP addresses for the Qwerk, Gateway, and Default Server for your network. Accept the default values for all the other options. Make sure to enter yes (y) when prompted whether to update the RedBoot non-volatile configuration.

In the example shown below, values you need to enter exactly as shown are shown in **black, bold type**. Values that you need to enter which might be different for your network are shown in **red, bold type**. You should use the defaults for all other values.

RedBoot> fconfig
Run script at boot: **true**
Boot script:
.. load -r -v -b 0x800000 ramdisk.gz
.. load -r -v -b 0x80000 zImage
.. exec -r 0x800000 -s 0x600000
Enter script, terminate with empty line
>> **fis load ramdisk**
>> **fis load zImage**
>> **exec -r 0x800000 -s 0x600000**
>>

Boot script timeout (1000ms resolution): 1

Use BOOTP for network configuration: false

Gateway IP address: **192.168.0.1**

Local IP address: **192.168.0.3**

Local IP address mask: 255.255.255.0

Default server IP address: **192.168.0.2**

DNS server IP address: **192.168.0.1**

Set eth0 network hardware address [MAC]: false

GDB connection port: 9000

Force console for special debug messages: false

Network debug at boot time: false

Update RedBoot non-volatile configuration - continue (y/n)? **y**

... Erase from 0x607c0000-0x607c1000: .

... Program from 0x01fde000-0x01fdf000 at 0x607c0000: .

RedBoot>

Now either enter reset or power-cycle the Qwerk so that your changes take effect. Either way, be ready to type CTRL-C to get back in to the RedBoot> prompt.

Loading the Linux Images

First, we'll load three images from the HTTP server (running on your machine) into the Qwerk's RAM.

Enter the following three load commands (the range of hex addresses displayed may be different):

RedBoot> **load -r -v -b 0x1000000 -m http /optfs.out**

|

Raw file loaded 0x01000000-0x011e099b, assumed entry at 0x01000000

RedBoot> **load -r -v -b 0x800000 -m http /ramdisk.gz**

\

Raw file loaded 0x00800000-0x0095c6d4, assumed entry at 0x00800000

RedBoot> **load -r -v -b 0x80000 -m http /zImage**

/

Raw file loaded 0x00080000-0x0019d12b, assumed entry at 0x00080000

RedBoot>

Now we'll initialize the flash filesystem. Answer yes (y) when it asks if it's ok initialize. The output should look like this (don't worry if the numbers in the output differ):

RedBoot> **fis init -f**

About to initialize [format] FLASH image system - continue (y/n)? **y**

*** Initialize FLASH Image System

... Erase from 0x60040000-0x60fc0000: .....

... Erase from 0x60fe0000-0x60fe0000:

... Erase from 0x61000000-0x61000000:

... Erase from 0x60fe0000-0x61000000: .

... Program from 0x01fdf000-0x01fff000 at 0x60fe0000: .

RedBoot>

Now we'll load the ramdisk and zImage images into flash (don't worry if the numbers in the output differ):

RedBoot> **fis create -b 0x800000 -l 0x300000 ramdisk**

... Erase from 0x60040000-0x60340000: .....

... Program from 0x00800000-0x00b00000 at 0x60040000: .....

... Erase from 0x60fe0000-0x61000000: .

... Program from 0x01fdf000-0x01fff000 at 0x60fe0000: .

RedBoot> **fis create -b 0x80000 -l 0x180000 -e 0x80000 zImage**

... Erase from 0x60340000-0x604c0000: .....

... Program from 0x00080000-0x00200000 at 0x60340000: .....

... Erase from 0x60fe0000-0x61000000: .

... Program from 0x01fdf000-0x01fff000 at 0x60fe0000: .

Now we'll load the optfs image into flash. The command to load the optfs image differs slightly depending on which Qwerk revision you have. The Qwerk revision is printed on the board between the analog inputs terminal and the top of the enclosure. It should read either "Qwerk 1.1A (c) 2006" or "Qwerk 1.2A (c) 2006". Determine which version you have and then use the appropriate command below to load the optfs image into flash.

If you have a revision 1.1 qwerk, use this command (don't worry if the numbers in the output differ):

RedBoot> **fis create -b 0x1000000 -l 0x300000 optfs**

... Erase from 0x603c0000-0x606c0000: .....

... Program from 0x01000000-0x01300000 at 0x603c0000: .....

... Erase from 0x607e0000-0x60800000: .

... Program from 0x01fdf000-0x01fff000 at 0x607e0000: .

RedBoot>

If you have a revision 1.2 qwerk, use this command (don't worry if the numbers in the output differ):

RedBoot> **fis create -b 0x1000000 -l 0xb00000 optfs**

... Erase from 0x604c0000-0x60fc0000: .....

... Program from 0x01000000-0x01b00000 at 0x604c0000: .....

... Erase from 0x60fe0000-0x61000000: .

... Program from 0x01fdf000-0x01fff000 at 0x60fe0000: .

RedBoot>

The difference is simply the value for the -l argument, which specifies the length. Revision 1.1 qwerks have 8 megs of flash RAM, but revision 1.2 qwerks have 16 megs of flash RAM. Using a value of 0xb00000 for the revision 1.2 qwerks enables the use of the extra 8 megs of RAM.

Testing the Linux Installation

Now either enter reset or power-cycle the Qwerk so that your changes take effect (but don't type CTRL-C, because we want the Qwerk to boot up into Linux). You should something similar to the following:

RedBoot> +Ethernet eth0: MAC address 00:dc:6c:7d:6b:25

IP: 192.168.0.3/255.255.255.0, Gateway: 192.168.0.1

Default server: 192.168.0.2, DNS server IP: 192.168.1.1

RedBoot(tm) bootstrap and debug environment [ROMRAM]

Non-certified release, version v2_0 - built 13:09:18, Mar 21 2006

Platform: Cirrus Logic EDB9302 Board (ARM920T) Rev A

Copyright (C) 2000, 2001, 2002, Red Hat, Inc.

RAM: 0x00000000-0x02000000, 0x00041fa8-0x01fdd000 available

FLASH: 0x60000000 - 0x60800000, 64 blocks of 0x00020000 bytes each.

== Executing boot script in 1.000 seconds - enter ^C to abort

RedBoot> fis load ramdisk

RedBoot> fis load zImage

RedBoot> exec -r 0x800000 -s 0x600000

Using base address 0x00080000 and length 0x00180000

Uncompressing Linux..... done, booting the kernel.

Wait a few seconds while the kernel boots. You should eventually see a message (shown below) that prompts you to type Enter to activate the console. Do so, and you should see:

Please press Enter to activate this console.

BusyBox v1.00 (2006.06.28-21:45+0000) Built-in shell (ash)

Enter 'help' for a list of built-in commands.

~ #

Congratulations! You successfully installed the firmware on the Qwerk, and it's ready to use!

Appendix E

Maintenance

This appendix lists the routine maintenance suggestions for Vertigo's up keeping. Most of these suggestions and timeframes are based on experience working with the system and assume daily use.

*Set screws on wheel hubs should be tightened weekly when in use. 1/8 in hex-key loosens the wheels retaining washer, then a .05 in hex-key fits the set screw. Be careful not to strip the hex or threads on these small screws.

*Keep an eye on battery life and how well they are charging.

*Watch for wheel erosion.

*Periodically check that all bolts and screws are securely fastened.

*Periodically check that none of the motors have shifted in their mounts so that all four wheels contact a flat surface evenly. To mount the motors evenly, make the

front face of the gear-head flush with the face of the front motor mount. Align them on a flat surface, applying pressure to keep both in contact with the surface as you tighten the mounts.